FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

JS Performance Certifier

André Gonçalo Correia Guedes de Gouveia Mota

PREPARAÇÃO DA DISSERTAÇÃO

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Rui Filipe Lima Maranhão de Abreu

25 de Fevereiro de 2016

JS Performance Certifier

André Gonçalo Correia Guedes de Gouveia Mota

Mestrado Integrado em Engenharia Informática e Computação

25 de Fevereiro de 2016

Resumo

O tema de dissertação centrado na área de *web profiling*, foi proposto pela empresa Glintt - Healthcare Solutions, S.A, com o objetivo de avaliar e certificar as suas aplicações web.

Atualmente, com a enorme diversidade de dispositivos eletrónicos, como os computadores e os dispositivos móveis, é quase impensável garantir a plena satisfação de eficiência de uma aplicação web para todos os utilizadores.

Estudos recentes revelam que contrariando o grau de exigência de anos transatos, os clientes dão grande importância à rapidez no carregamento de uma página web tanto em computadores como em dispositivos móveis, o que tornou o desempenho um fator essencial para as aplicações. Esta necessidade despoletou o aparecimento de um novo conceito designado *web application profiling*. Como tal, é necessário rever o paradigma que afirma que a performance não é um dos aspetos mais importantes do desenvolvimento de um projeto web.

O objetivo primordial de aumentar a performance é minimizar o *delay* para o utilizador, com a finalidade de garantir uma melhor experiência de navegação. Este *delay* é importante durante todo o tempo de utilização da aplicação, desde o momento de carregamento em que o utilizador introduz o URL, até ao momento em que a página se encontra totalmente carregada, assim como durante a sua utilização. Esta vai ser uma das métricas utilizadas na dissertação, juntamente com a quantidade de memória e processador usado. A utilização de código *Client-Side* torna as páginas mais dinâmicas e responsivas, contudo pode induzir um maior *delay* para o utilizador ao tornar aplicação menos eficiente.

Inicialmente é apresentado um estado de arte na área da performance web. O projeto final visa apresentar um protótipo dividido em três partes interligadas, cada uma responsável por testar componentes distintos: uma primeira fase, o objetivo é analisar e reportar o tempo de carregamento da aplicação; na fase seguinte, o objetivo é acompanhar a aplicação e proceder à avaliação minuciosa da quantidade de memória e processador utilizado; o último momento consiste em testar e avaliar criteriosamente o código *Javascript* com o propósito de detetar inconsistências e apresentar ao utilizador uma solução de forma a contornar e, se possível, solucionar o problema.

Keywords: Performance, Javascript, HTML5, Web2.0, Métricas, Profiling

Abstract

The dissertation topic focused on web application profiling area was proposed by the company Glintt - Healthcare Solutions, SA, in order to evaluate and certify their web applications. Currently, with the huge variety of electronic devices such as computers and mobile devices, it is almost unthinkable to guarantee full satisfaction and efficiency of a web application for all users. Recent studies reveal that contrary to the past years, customers attach great importance to speed loading of a web page both on computers and on mobile devices, which made the performance a key factor for applications. This need triggered the appearance of a new concept called web application profiling. As such, it is necessary to review the paradigm that says the performance is not one of the most important aspects of developing a web project. The primary goal of increasing performance is to minimize the delay for the user, in order to ensure a better browsing experience. This delay is important during the entire time of the application use, since the moment of loading, when the user enters the URL, to the time when the page is fully loaded, as during use. This will be one of the metrics used in the dissertation, together with the use of memory and processor. The use of client-side code makes the pages more dynamic and responsive, yet it can induce a greater delay for the user by making the application less efficient. Initially it will be presented a state of the art in web performance area. The final project aims to present a prototype divided into three interconnected parts, each responsible for testing distinct components: on the first phase, the goal is to analyze and report the ammount of time it takes for the application to load; in the next phase, the objective is to monitor the application to assess the ammount of memory and processor used; lastly is will test and evaluate carefully the Javascript code in order to detect inconsistencies and present the user a solution in order to bypass and, if possible, resolve the issue.

Keywords: Performance, Javascript, HTML5, Web2.0, Metrics, Profiling

Agradecimentos

Este estudo é o resultado de um empenho individual com o contributo de várias pessoas que o tornaram possível e que sem dúvida foram cruciais para concluir esta etapa, que representa uma importante fase na minha vida pessoal e profissional. Desta forma quero agradecer a todos os que estiveram presentes, em particular:

Ao meu orientador, Prof. Dr. Rui Maranhão, pela sua total disponibilidade e contributo, pelas opiniões e criticas construtivas ao longo da realização da tese.

À FEUP e Glintt HS, por me proporcionarem as condições necessárias para a realização deste trabalho.

Aos meus colegas da Glintt HS, que desde o primeiro dia, me proporcionaram um bom ambiente de trabalho.

Em especial, à minha família e namorada, por serem um exemplo a seguir e pelo apoio incondicional sempre que eu precisei, sem eles nenhum obstáculo seria fácil de ultrapassar. Obrigada pela confiança depositada em mim e por todos os sacrifícios feitos para que fosse possível eu chegar aqui. A eles dedico este trabalho!

André Mota

"It is impossible to live without failing at something, unless you live so cautiously that you might as well not have lived at all, in which case you have failed by default."

J.K. Rowling

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

AJAX	Asynchronous Javascript and XML
API	Application Programming Interface
CSS	Cascading Style Sheets
DNS	Domain Name Server
DOM	Document Object Model
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
OS	Operating System
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
URL	Universal Record Locator
W3C	World Wide Web Consortium
WWW	*World Wide Web*
XHR	XML HTTP Request

Capítulo 1

Introdução

Vivemos num mundo global cada vez mais competitivo onde a inovação é um fator primordial. A ascensão da Web 2.0 conferiu uma mudança na forma como é executada a interação entre utilizador e aplicação web, o que provocou um aumento exponencial no tamanho e complexidade das aplicações [O'r07].

1.1 Contexto/Enquadramento

Esta dissertação de mestrado foi proposta pela empresa Glintt - Healthcare Solutions, S.A. A Glintt - Healthcare Solutions é uma empresa integrada no grupo Glintt, que conta com mais de 20 anos de experiência no setor da saúde. Dedica-se à prestação de serviços na área de IT, tendo como objetivo principal o desenvolvimento de projetos na área da Gestão da Saúde. Neste momento, já marca a sua presença em mais de 200 hospitais e clínicas do país.

Com a crescente melhoria na velocidade da internet, a performance das aplicações tornou-se um fator extremamente importante, e em algumas situações essencial na competitividade entre empresas [Rem15]. Essa performance pode ser resumida ao tempo de carregamento da aplicação e seus componentes à vista do utilizador, bem como ao tempo de resposta durante a sua utilização. A performance será resultante da execução de código do lado do servidor e também do lado do cliente, sendo este último o alvo de estudo neste projeto [Kie10].

> "Ever since Steve Souders' High Performance Web Sites book, interest in making web sites load faster has been increasing. More and more big companies with a strong web presence are paying attention to page loading performance: the well-known ones such as Microsoft, Yahoo, Google, but also big companies that are not technology companies such as Amazon, White Pages, Shopzilla, Edmunds, Netflix..." [Lee11, chap. Context]

Fazer *profiling* de aplicações web é uma tarefa bastante complexa comparada com as restantes áreas de *software profiling*, devido ao facto de serem multicamadas. Este aumento de complexidade resulta do défice de performance ter origem tanto no seu mau funcionamento, como na sua instalação, configuração, ou mesmo na comunicação entre camadas.

O código *Client-Side* tem vindo a afirmar-se no mundo das tecnologias não só pelo aumento das velocidades de internet, mas também para satisfação do conceito visual dos utilizadores. Este código é executado na sua totalidade no browser, o que provoca um aumento acrescido no carregamento e delay da aplicação web para o utilizador [Kie10][Sou08].

A presente dissertação foi supervisionada pelo Eng. Francisco Correia, da Glintt HS, que acompanhou o seu desenvolvimento e, academicamente acompanhada, na FEUP, pelo Prof. Dr. Rui Maranhão.

1.2 Projeto

A solução apresentada está dividida em 6 componentes. O primeiro é uma biblioteca Javascript que responde ao primeiro objetivo: avaliar a experiência do cliente ao interagir com a aplicação. Os dois componentes seguintes são utilizados para acompanhar a aplicação e monitorizar a utilização de memória e processador. O quarto componente é uma Dashboard, que tem a finalidade de oferecer à empresa uma aplicação web onde é possível analisar os dados de todas as interações entre os seus clientes e aplicações num só local. O quinto é um servidor que proporciona um elo de comunicação entre todos os componentes. O último representa toda a documentação interna e também esta dissertação.

1.3 Motivação e Objetivos

O nível de satisfação exigido por clientes de empresas prestigiadas como é o exemplo da Glintt HS é cada vez mais elevado.

Esta dissertação no campo de *web profiling* pode trazer um avanço considerável para o futuro desta área e representar uma vantagem para os diversos atores de aplicações web de acordo com o seguinte:

- Para os *developers* resulta numa diminuição do tempo necessário à análise e *debugging*[1] [AZGvG09][AZV07] das aplicações para obter os problemas relativos ao défice de performance da aplicação, pois essa análise será feita automaticamente e apresentada aos mesmos [GCA13][Gem][CAFD13].

- Em relação aos clientes, os problemas relativos ao tempo de carregamento serão mais facilmente identificados e ultrapassados resultando numa maior satisfação e melhor experiência de navegação.

[1] Processo de procura e redução de defeitos numa aplicação de software.

A redução de tempo de desenvolvimento e *debugging* por parte dos *developers* aliada a uma maior satisfação dos clientes podem representar uma vantagem competitiva para a empresa Glintt - Healthcare Solutions, S.A. Com base nesta premissa foram traçados um conjunto de objetivos para todo o processo de investigação e desenvolvimento:

- Analisar o estado de arte em *web profiling* e as tecnologias existentes.
- Definir a viabilidade e, se possível, desenvolver uma solução que permita automatizar o processo de análise de *performance*.
- Integrar a solução final com as tecnologias da empresa para obter um número aceitável de casos práticos para avaliação.

1.4 Estrutura da Dissertação

Para além da introdução, esta dissertação contém mais 3 capítulos. No Capítulo 2, é descrito o estado da arte e a apresentação de trabalhos relacionados. No Capítulo 3, é apresentado o problema que esta dissertação visou solucionar, mostrando todo o sistema da solução JS Performance Certifier juntamente com a sua arquitetura e requisitos. No Capítulo 4, são expostos os detalhes do desenvolvimento onde são descritas as tecnologias utilizadas, problemas identificados e soluções implementadas. No Capítulo 5, é apresentada uma descrição sucinta do que foi feito na área e o plano de trabalho para o desenvolvimento da solução final.

Capítulo 2

Revisão Bibliográfica

O código do lado do cliente, (em inglês *Client-Side code* ou *Front-End code*), é responsável na grande maioria dos casos pela numerosa percentagem de tempo de carregamento e utilização da aplicação. Assim, a sua análise e aperfeiçoamento são a melhor opção para aumentar a *performance* [Sou08].

Este capítulo é dedicado à descrição do estado de arte complementado com estudos anteriormente publicados. A última parte aborda a apresentação das diferentes técnicas de análise e métricas utilizadas para avaliar a performance de uma aplicação web.

2.1 Tempos de Carregamento

Uma das métricas mais importantes na análise de performance de uma aplicação web é o tempo de carregamento. A avaliação deste tempo indica se uma aplicação está ou não com défice de performance.

Com o tempo de espera, mesmo as pessoas mais pacientes acabam por abandonar as aplicações web após um longo período [GHMP02], é por isto que a performance se tornou um aspeto importantíssimo [Bru09].

Para fazer uma análise do tempo de carregamento é necessário saber qual o processo realizado desde o momento inicial, em que o utilizador clica no URL, até ao momento final, que acontece quando a página está totalmente carregada, cujos momentos se passa a enunciar [Mee13] [SB01]:

1. O utilizador clica no URL ou introduz o mesmo manualmente.
2. De seguida é feito o DNS *lookup*[1] para atribuir um IP ao URL introduzido.
3. Após o servidor de DNS responder com um IP é feito uma conexão TCP com o servidor da aplicação.
4. Então, é enviado um pedido ao servidor para obter o HTML da página.

[1] Pedido que traduz o domínio em um endereço de IP.

5. O pedido fica em espera na fila, para a aplicação web.
6. O servidor processa o pedido e envia a resposta de volta pela rede, para o browser do utilizador (*First Byte*[2]).
7. O browser do utilizador faz *parsing* da resposta do servidor e começa a fazer render (*DOM Ready*[3]).

Agora, é pertinente concluir que o carregamento depende de diversos fatores, tornando a performance dependente não só da qualidade de estrutura e implementação, mas também da velocidade da conexão do utilizador bem como na configuração do próprio dispositivo.

Todo o gasto temporal executado do lado do servidor vai ser ignorado pelo facto de já existirem variadas técnicas para analisar e corrigir esse tempo e por este corresponder a uma pequena percentagem.

Table 1. Percentage of time spent on the backend.

Web Site	Empty Cache	Primed Cache
http://www.aol.com/	3%	3%
http://www.ebay.com/	5%	19%
http://www.facebook.com/	5%	19%
http://www.google.com/search?q=flowers	53%	100%
http://search.live.com/results.aspx?q=flowers	33%	100%
http://www.msn.com/	2%	6%
http://www.myspace.com/	2%	2%
http://en.wikipedia.org/wiki/Flowers	6%	9%
http://www.yahoo.com/	3%	4%
http://www.youtube.com/	2%	3%

Figura 2.1: Percentagem de tempo utilizado em *back-end* [Sou08].

Em seguida, é feita uma análise mais aprofundada sobre as diferentes técnicas e API's usadas para obter os tempos de carregamento.

2.1.1 Navigation Timming

Uma das técnicas utilizadas para avaliar o desempenho do carregamento de uma aplicação web, é recolher informação sobre o chamado *"user percieved time"* que corresponde ao tempo, desde o momento em que o utilizador decide entrar na página, até à fase em que o conteúdo está totalmente carregado e apresentado.

[2]Designação dada ao momento em que o browser recebe a informação do servidor.
[3]Processo de desenho realizado pelo browser para mostrar o conteúdo da página ao utilizador.

Até há alguns anos atrás, os *developers* das aplicações web, para medir a performance de uma página, valiam-se de um excerto de código Javascript no inicio do documento HTML similar ao visível na Listing 2.1.

```
<!doctype html>
<html>
   <head>
      <script>
         var start = new Date().getTime();
         window.addEventListener('load', function() {
            var end = new Date().getTime();
            console.log('Tempo de carregamento (em milisegundos): ' + (end - start)
                );
         });
      </script>
   </head>
   <body>
   </body>
</html>
```

Listing 2.1: Exemplo de código Javascript para obter o tempo de carregamento da página.

No entanto, existem alguns problemas ao utilizar este excerto, nomeadamente o tempo fornecido pelo objeto *Date*, que é impreciso e dependente de ajustes do relógio, o que pode induzir o utilizador em erro. Além disso, este objeto apenas nos fornece informação, a partir do momento em que está a ser feita a renderização, ou seja, todo o tempo que envolve a comunicação e obtenção do HTML do servidor não é tido em conta [Mee13].

Para contornar este problema da ineficiência do objeto Date do Javascript e dos ajustes do relógio, o W3C propôs no ano de 2012 a *Navigation Timing API* [Net15b] [W3C13] [Mee13], com o propósito de conceder a possibilidade para os *developers* analisarem de melhor forma, o desempenho das suas aplicações. A *Navigation Timing API*, utiliza um novo formato de tempo presente nos *browsers* mais recentes de nome *High Resolution Time* [Iri12] que dispõe de resoluções ao nível de micro segundos, o que é mil vezes mais preciso do que o método anterior. Ao inverso do objeto Date, esta API fornece informação sobre os tempos relacionados com o DNS lookup, conexão TCP, tempo de carregamento do DOM, entre outras.

	IE (Desktop and Mobile)	Chrome (Desktop and Android)	Firefox (Desktop and Android)	Safari (Desktop and Mobile)
Navigation Timing	9+	6+	7+	--
Resource Timing	10+	26+	--	--
requestAnimationFrame	10+	10+	4+	6+
High Resolution Time	10+	21+	15+	--

Figura 2.2: Suporte de browsers às APIs apresentadas [Mee13].

Como a Navigation Timing API é reconhecida pelo W3C, existe bastante suporte a nível de versões dos browsers mais utilizados.

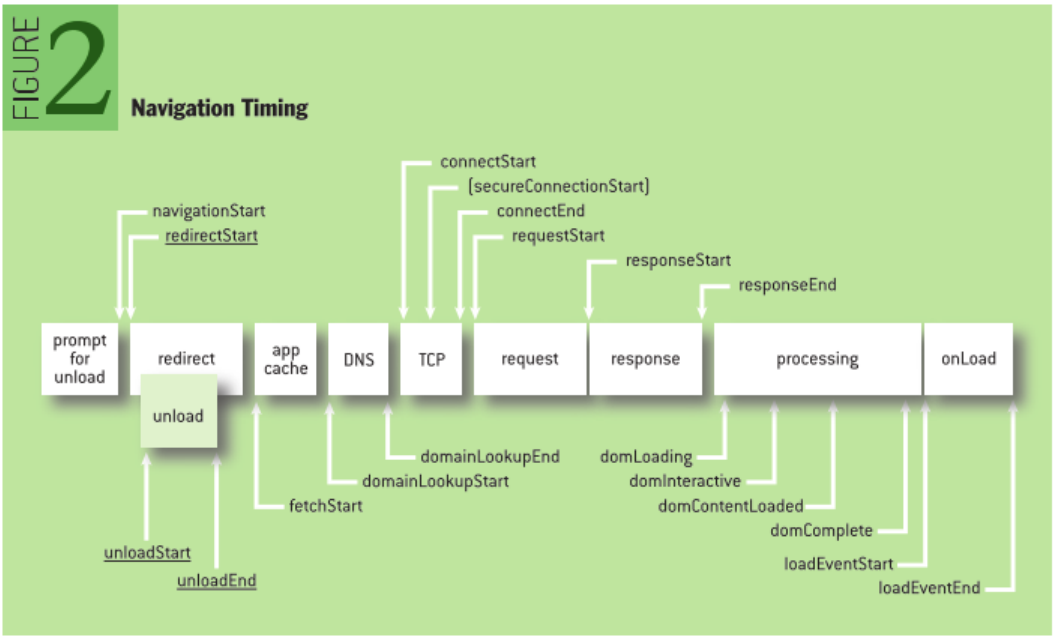

Figura 2.3: Variáveis fornecidas pela Navigation Timing API [Mee13].

A API recolhe informação sobre diversos eventos relacionados com o processo de carregamento da página, registando os mesmos no objeto *window.performance.timming*. É pertinente nesta fase apresentar uma lista de todos os eventos fornecidos por esta API e o seu significado [Gri14] [W3C13]. Na Figura 2.3, é possível visualizar a ordem dos mesmos.

navigationStart Esta variável representa o tempo após ser iniciada a navegação.

unloadEventStart Tempo antes do evento *"unload"* da página anterior ser executado, caso não exista esse documento, o valor é 0.

unloadEventEnd Mesmo significado da função anterior, mas representa o valor após esse evento.

redirectStart Em caso de redirecionamento da página, esta variável armazena o tempo de início do pedido.

redirectEnd Tempo após receber o último byte do último redirect.

fetchStart *"fetchStart"* representa o tempo antes de o servidor começar a procurar pelo URL.

domainLookupStart Este valor corresponde ao início do *DNS lookup* se necessário, caso contrário tem o mesmo valor que *fetchStart*.

domainLookupEnd Como o nome indica, representa o tempo após finalização do *DNS lookup*.

connectStart Representa o tempo antes do *browser* estabelecer a conexão com o servidor.

connectEnd Tempo após fim de conexão. É de notar que em caso de ser um recurso em cache[4] ou local, o valor das variáveis *connectStart* e *connectEnd* tomam o mesmo valor que *domainLookupEnd*

secureConnectionStart No caso de ser utilizado o protocolo HTTPS[5], este valor representa o tempo que demora a negociação SSL.

requestStart Inicio do pedido ao servidor.

responseStart Corresponde ao tempo, após o *browser* receber a resposta do servidor (*First Byte*).

responseEnd Momento imediatamente após receber o último byte do servidor.

domLoading Este valor representa o início de renderização do HTML por parte do *browser*, muitas vezes referido como *Critical Rendering Path* [Gri14].

domInteractive Momento anterior à flag *document.readyState* ser colocada como *"interactive"*.

domContentLoadedEventStart Representa o tempo antes do evento *DomContentLoaded* ser executado.

domContentLoadedEventEnd Momento após o evento *DomContentLoaded* ser executado.

domComplete Tempo imediatamente antes do momento final de renderização.

loadEventStart Representa o tempo antes do evento *window.onload* ser executado.

loadEventEnd Tempo após o evento *window.onload* ser executado.

Uma outra funcionalidade é o facto de esta API revelar também de que forma o utilizador chegou à página que está a ser analisada, através do objeto *window.performance.navigation* que contém duas variáveis, *type* e *redirectCount*.

Em relação à variável *redirectCount*, ela armazena o número de redirects[6] efetuados até ao utilizador entrar na página, ou zero em caso de não existir nenhum. A variável *type* pode tomar quatro valores diferentes:

TYPE_NAVIGATE - valor "0", caso o utilizador entre na página através da introdução do URL ou hiperligação.

TYPE_RELOAD - a variável toma o valor "1"se a navegação resultar de um refresh ou reload[7].

TYPE_BACK_FORWARD - retroceder ou avançar no histórico resulta no valor "2".

[4] Processo que armazena data sobre o recurso, localmente, na medida em os próximos carregamentos sejam executados mais rapidamente.

[5] protocolo HTTP sobre uma camada adicional de segurança que usa o protocolo SSL.

[6] Processo de redirecionamento de uma página para um domínio diferente.

[7] Ao executar o refresh ou reload de uma página, esta é carregada novamente

TYPE_RESERVED - por ultimo o valor "255", menos usual, em caso de qualquer outro tipo de navegação.

Em suma, esta API é extremamente robusta e completa oferecendo também a possibilidade de obter o tempo gasto pela conexão com o servidor, desde a resolução do DNS, estabelecer a ligação TCP, entre outras, o que não era possível anteriormente. O passo final é organizar a informação e demonstrá-la da melhor forma, diretamente para o utilizador no *browser*, ou enviando informação sobre os tempos por XHR para um servidor remoto e posteriormente retirar informação relevante [Net15b].

2.1.2 Resource Timming

Até ao momento foi analisado o método de recolha dos tempos relativos ao carregamento de uma página. Mas, e se uma página estiver com problemas de performance? Onde devemos procurar? A resposta a todas estas questões, pode residir na API que será apresentada a seguir.

A *Resource Timing* API [Gri13] [W3C15] fornece a possibilidade de obter informação sobre todos os recursos carregados por uma página, desde imagens, ficheiros Javascript, *CSS* ou mesmo bibliotecas externas [GG14a]. Esta API pode ser consultada também através do objeto *window.performance* e em seguida o acesso a todos os recursos carregados é feito através da chamada *"getEntriesByType('resource')"* [W3C15].

Assim é possível determinar se a razão do défice de *performance*, é devido a um ficheiro carregado, uma biblioteca externa, entre outros, fornecendo um maior poder de análise aos *developers* da aplicação [Gri13].

```
1  if ( !('performance' in window) ||
2       !('getEntriesByType' in window.performance) ||
3       !(window.performance.getEntriesByType('resource') instanceof Array)
4     ) {
5    return false;
6  } else {
7     window.addEventListener('load', function() {
8        var resources = window.performance.getEntriesByType('resource');
9     });
10 }
```

Listing 2.2: Exemplo de código Javascript para obter os recursos carregados por uma página [W3C15].

Na Figura 2.4, é exemplificado o resultado do primeiro objeto do *array*[8] [Deva].

Existem no entanto algumas limitações. Em primeiro lugar, não é possível obter os tempos de bibliotecas externas caso as mesmas não implementem o HTTP header *"(Timing-Allow-Origin: *)"* [W3C15], felizmente os principais fornecedores como Google e Facebook já o fazem, para ser

[8]Estrutura de dados, destinada a armazenar varias variáveis, do mesmo tipo, num só objeto.

```
> window.performance.getEntries()[0]
▼ PerformanceResourceTiming {responseEnd: 618.0039999962901, responseSta
    connectEnd: 532.8390000003856
    connectStart: 449.83900000038557
    domainLookupEnd: 449.83900000038557
    domainLookupStart: 449.83900000038557
    duration: 170.40399999677902
    entryType: "resource"
    fetchStart: 447.59999999951106
    initiatorType: "link"
    name: "http://www.w3.org/StyleSheets/TR/W3C-CR.css"
    redirectEnd: 0
    redirectStart: 0
    requestStart: 532.8390000003856
    responseEnd: 618.0039999962901
    responseStart: 617.8390000003856
    secureConnectionStart: 0
    startTime: 447.59999999951106
  ▶ __proto__: PerformanceResourceTiming
```

Figura 2.4: Exemplo do primeiro recurso carregado [Deva].

possível avaliar a latência da inclusão das suas bibliotecas nas páginas web. Em segundo, para os recursos em *cache* como imagens, ficheiros CSS, Javascript, entre outros, a sua latência vai ser zero, visto que esta API apenas avalia pedidos HTTP e não pedidos ao OS e na realidade, nos recursos em cache a latência é extremamente baixa, por isso a sua análise torna-se redundante [Gri13]. Por fim, como esta API ainda está nos primórdios do seu desenvolvimento, o suporte a nível de *browsers* não é o desejável como se pode verificar na Figura 2.2. Apesar disso, o facto de principais marcas como o Google Chrome, Firefox (neste momento fornece suporte a esta API a partir da versão 37), Internet Explorer e Opera (não visível na Figura 2.2, mas existe suporte para versões posteriores à 15) a suportarem, faz com que esta API seja uma proposta passível a ser utilizada atualmente.

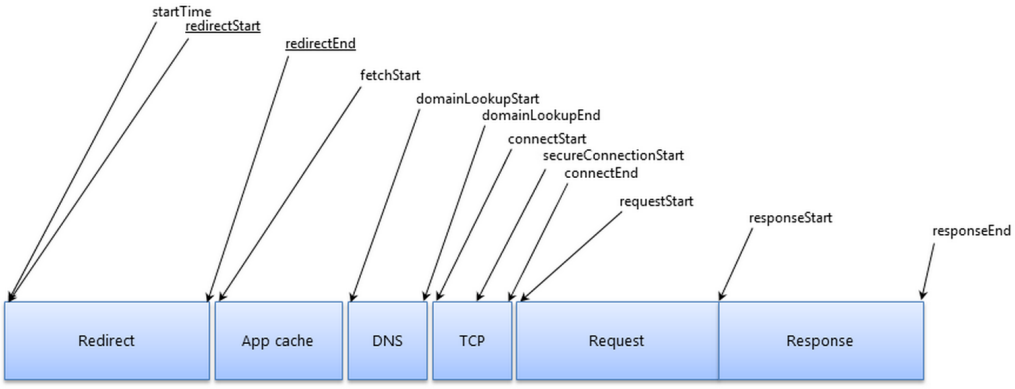

Figura 2.5: Variáveis fornecidas pela Resource Timing API [W3C15].

Para cada recurso recolhido por esta API é criado um objeto do tipo *PerformanceResourceTiming* que dispõe da seguinte informação [W3C15]:

name - o URL do recurso.

entryType - este valor é sempre igual a "resource".

startTime - o tempo inicial do pedido do recurso ao servidor.

responseEnd - o tempo após o recurso ser processado e carregado.

duration - diferencça entre responseEnd e startTime.

initiatorType - esta variável corresponde ao elemento da página responsável pela chamada (script, img, css, entre outras).

Mesmo estando numa fase inicial, a *Resource Timing* API tem bastante potencial. Cada vez mais as principais marcas de *browsers* procuram oferecer o seu suporte para versões menos recentes, o que facilita a perceção e análise do que hoje é cada vez mais, um fator essencial no desenvolvimento e manutenção de aplicações web.

2.1.3 New Relic

A conjugação das duas APIs apresentadas anteriormente possibilita a recolha de uma quantidade de informação, que pode ser extremamente benéfica para o utilizador. Um bom exemplo dessa utilização é o *New Relic Browser* [SYA+14].

O *New Relic Browser*, é um serviço de monitorização de performance que executa diretamente no dispositivo e *browser* do cliente. Este procura reunir dados, para reportar com precisão como é a experiência de navegação para o utilizador [dSN14] [YL12].

Por cada página carregada são registados os seguintes eventos [9]:

- Tempo utilizado no Front-End.
- Código e eventos executados no browser.
- Tempo utilizado no Back-End.
- Localização geográfica.
- Tipo de browser, versão e Sistema Operativo.

A informação é armazenada e organizada por visitas, sessão e localização e possibilita identificar inconsistências como por exemplo, problemas com uma versão de um browser, ou mesmo diminuição de performance numa zona geográfica específica.

[9] https://docs.newrelic.com/docs/browser/new-relic-browser/welcome-new-relic-browser/new-relic-browser

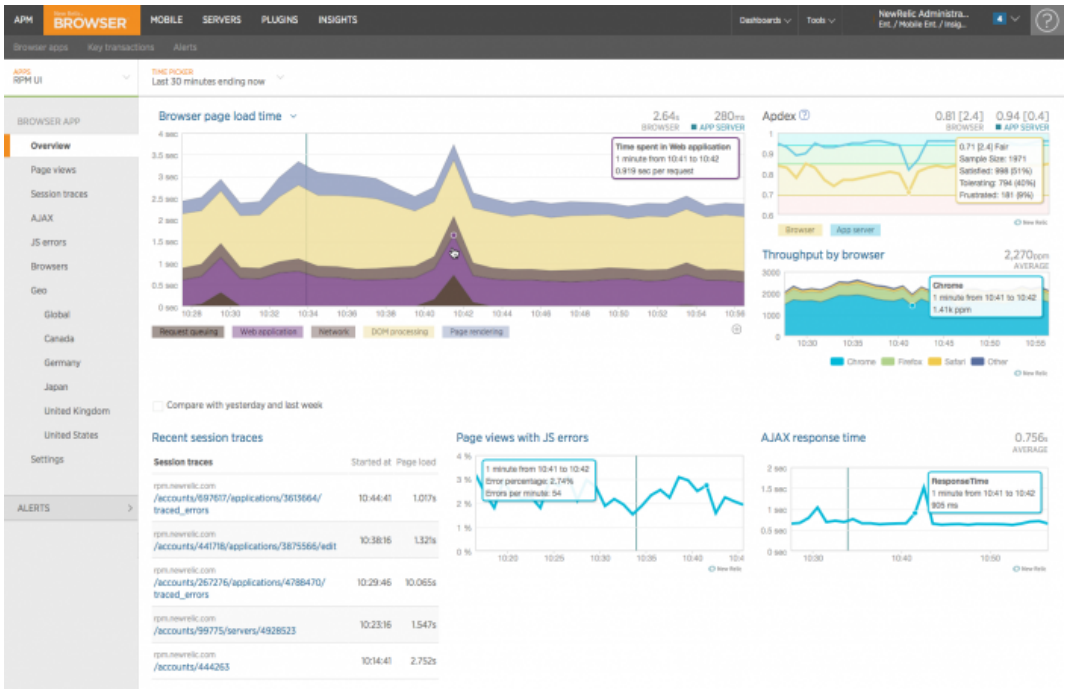

Figura 2.6: Overview dashboard.

A Figura 2.6[10] representa a *dashboard*[11] com o sumário da informação recolhida pelo *New Relic*. É possível obter informação mais detalhada, navegando pelo menu lateral para os seguintes casos específicos[12]:

Page Views Fornece informação sobre o desempenho das páginas mais visitadas.

Session Traces Contém dados sobre todo o ciclo de vida de uma página, desde chamadas AJAX, erros, carregamento de recursos, entre outros. Apenas disponível na versão PRO.

Ajax Nesta página são apresentadas todas as chamadas AJAX efetuadas, com o respetivo tempo de execução e código de resposta. Esta funcionalidade precisa ser ativada nas definições da aplicação. Apenas disponível na versão PRO.

Javascript Errors Página onde são apresentados todos os erros que ocorrem durante a utilização da aplicação. Apenas disponível na versão PRO.

Browsers Esta secção permite analisar o desempenho da aplicação pelos diferentes *browsers* do mercado.

Geo Funcionalidade que organiza os dados por localização geográfica onde é possível identificar a experiência de utilização da aplicação em diferentes países ou cidades.

[10] https://docs.newrelic.com/docs/browser/new-relic-browser/getting-started/browser-overview-website-performance-glance

[11] interface de utilizador, onde a informação está organizada e apresentada de forma a ser facilmente analisada.

[12] https://docs.newrelic.com/docs/browser/new-relic-browser/getting-started/browser-overview-website-performance-glance

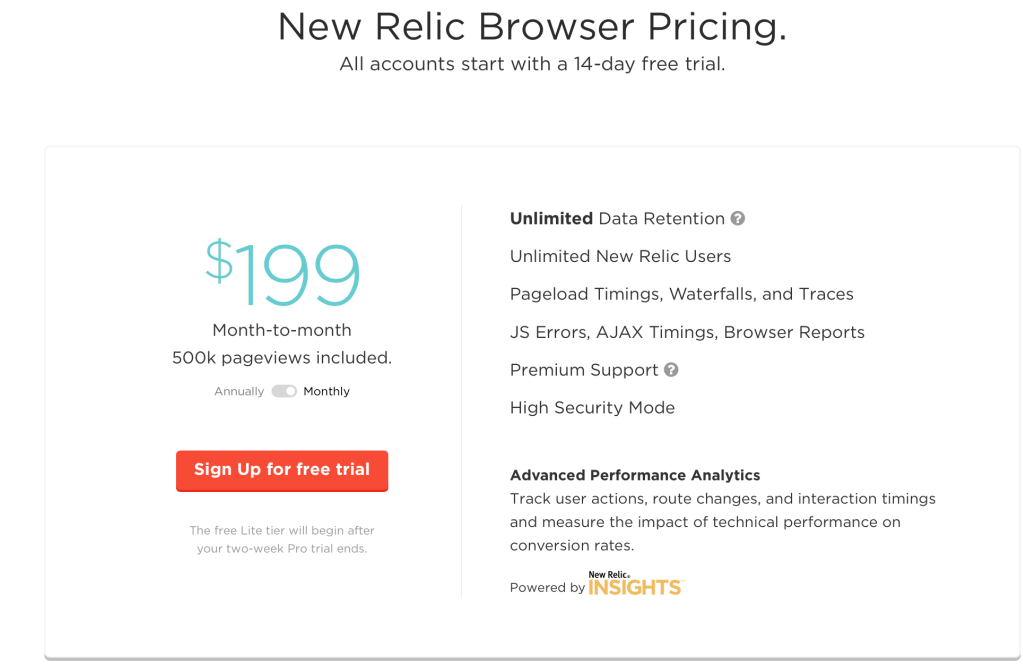

Figura 2.7: New Relic Browser - preço de subscrição.

Esta ferramenta está disponível nas versões *"standart"* e *"pro"* em que esta necessita de uma licença mensal mínima de 199$ até 500.000 visitas, mas com acesso a mais dados e funcionalidades [dSN14].

Uma outra vantagem deste serviço, é o facto de possibilitar a personalização da *Dashboard* e criação alertas para determinadas métricas.

2.1.4 Azure Application Insights

Um outro serviço similar ao apresentado na secção anterior, é o *Azure Application Insights* da *Microsoft*. Este serviço monitoriza não só o lado de cliente, mas também o lado do servidor, oferecendo compatibilidade para diversas plataformas [Wil15d] como por exemplo:

- Servidores com ASP.NET alojados na *cloud* Azure ou servidor IIS.
- Serviços Azure Cloud.
- Servidores J2EE.
- Páginas Web: HTML+JavaScript.
- Aplicações Windows de cliente e servidor.
- Outras Plataformas - Node.js, PHP, Python, Ruby, Joomla, SharePoint, WordPress.

Apesar das vantagens imediatamente visíveis, este serviço necessita obrigatoriamente de uma licença para ser utilizado e não está disponível na Europa Ocidental, como é possivel verificar na Figura 2.9.


REGIÃO: Centro dos E.U.A.

MOEDA: Euro (€)

	GRATUITO	STANDARD	PREMIUM
Número de Anfitriões e Dispositivos	Ilimitado	Ilimitado	Ilimitado
Dados de sessão (por mês)	Ilimitado	Ilimitado	Ilimitado
Outros pontos de dados (por mês)	5 milhão	15 milhão	50 milhão
Exportação de dados contínua	--	✓	✓
Dados Não Processados	7 dias	15 dias	30 dias
Dados Agregados	13 meses	Ilimitado	Ilimitado
Preços Mensais	€0	€20.67	€83.91
Pontos de Dados Adicionais	N/A	€1.4758/milhão	€1.6866/milhão

Dados de sessão – inclui contagem de sessões, utilizadores, geografia e outros dados de dispositivo e ambiente.

Outros pontos de dados – inclui todos os tipos de dados (exceto dados de sessão), como, por exemplo, eventos personalizados, dependências, exceções, métrica personalizada, carregamentos de página, visualizações de página, contadores de desempenho, pedidos e dados de rastreio.

Exportação de dados contínua – envia os dados telemétricos completos como um fluxo contínuo para uma base de dados à escolha.

Dados não processados – inclui acesso a todos os pontos de dados telemétricos recolhidos pelo Application Insights.

Dados agregados – inclui acesso a pontos de dados telemétricos agregados por hora/dia

Preço Mensal – a Faturação é rateada por hora. O preço acima baseia-se em 744 horas por mês de calendário.

Figura 2.8: Azure Application Insights - informação de licença.

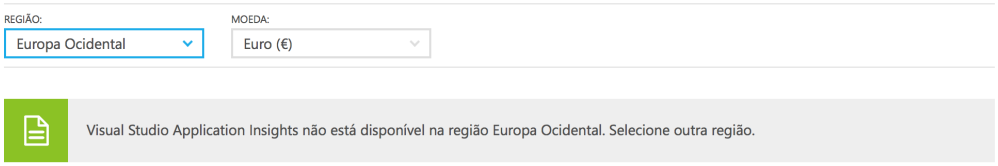

Figura 2.9: Azure Application Insights - restrição regional.

Como nesta dissertação apenas é dado ênfase à performance relacionada com a experiência para o utilizador, será somente analisado o componente deste serviço destinado a páginas web [Wil15b].

O tipo de dados recolhido por este serviço é identico ao New Relic, entre eles:

- Informação sobre os utilizadores.
- Sessões customizáveis.
- Páginas visualizadas.
- Tipo de Browser e Sistema Operativo.
- Informação geográfica.
- Erros ocorridos durante a sessão.
- Comunicações com servidores.
- Eventos customizados.

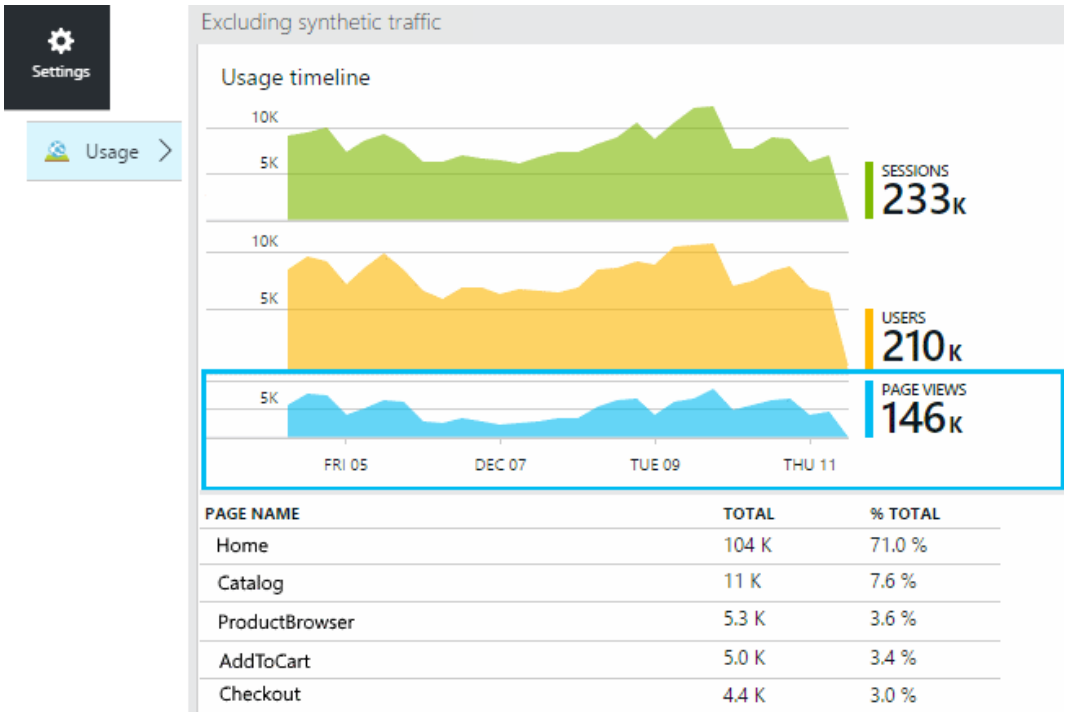

Figura 2.10: Application Insights for Javascript.[Wil15a]

O Application Insights oferece diversas opções de manipulação dos dados. Não foi possível analisar todas as funcionalidades visto ser necessário uma licença para o conseguir explorar aprofundadamente.

Resumidamente, este serviço oferece informação para dois tipos de análise. Em primeiro lugar, dados acerca de utilizadores, sessões e páginas. Por fim, dados enviados por telemetria em que é necessário introduzir pedaços de código na aplicação que se deseja monitorizar.

De seguida são apresentadas todas as opções de organização e apresentação da informação recolhida, presentes neste serviço.

Visão Geral

O principal propósito é dar a conhecer como é que a aplicação está a ser utilizada. A monitorização é realizada em tempo real, organizando os dados necessários para identificar ineficiências e problemas de performance.

Com a informação reunida é imediatamente identificável a popularidade da aplicação, as regiões onde esta é mais acedida ou até que tipo de OS é mais utilizado. É apresentado de seguida na Figura 2.11, um exemplo de informação recolhida de uma aplicação monitorizada por este serviço.

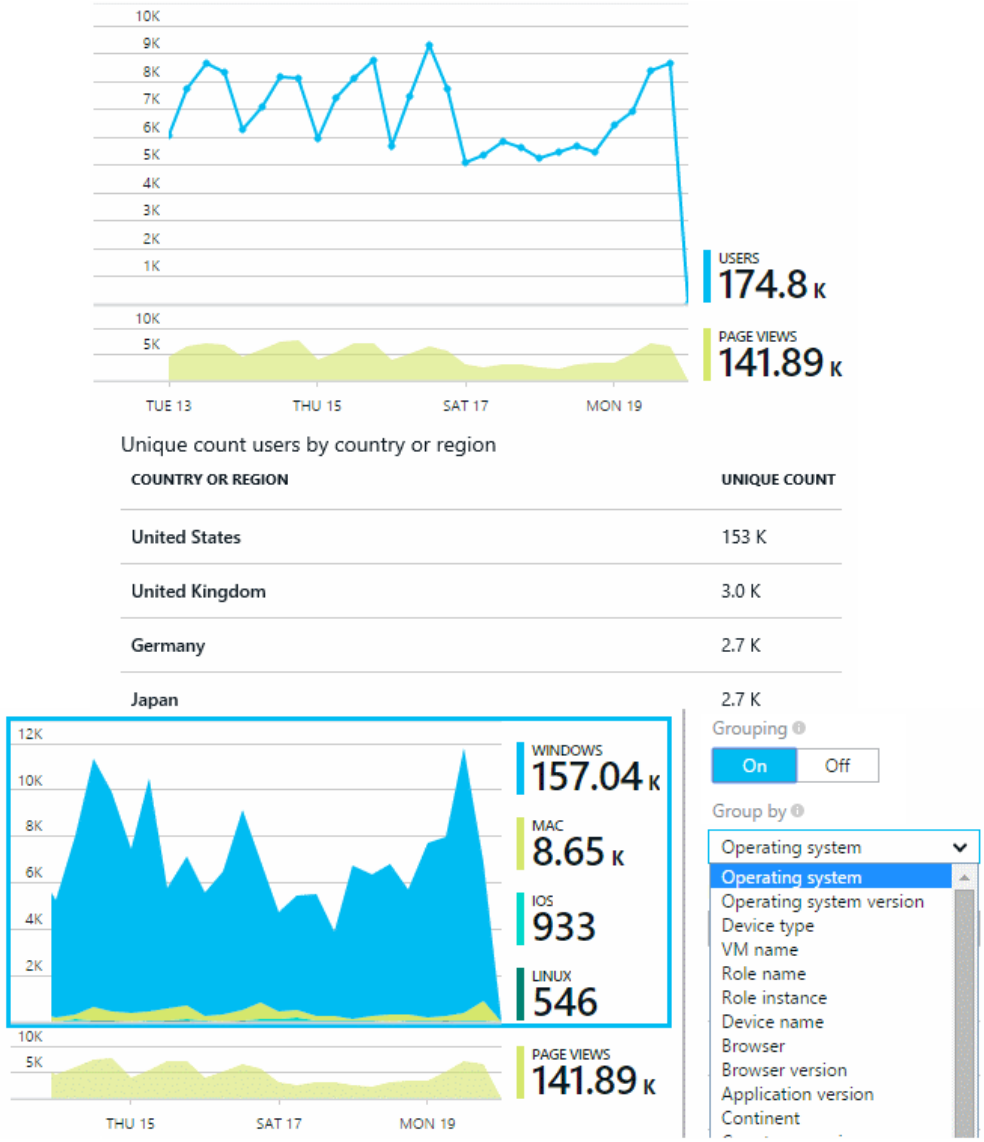

Figura 2.11: Application Insights for Javascript.[Wil15a]

Métricas

As métricas no Application Insights, representam valores e eventos que são enviados por telemetria pela aplicação monitorizada. Existe também a opção de criar alertas para determinadas métricas e notificar os responsáveis da aplicação por correio eletrónico.

Diagnostic Search

Neste local, o utilizador consegue observar pormenorizadamente cada métrica. Pode realizar pesquisas individuais de exceções, eventos, chamadas HTTP, entre outros.

É também exequível filtrar a informação por tipo, propriedade, valor ou dados específicos.

Eventos Customizáveis

A criação de eventos customizados, é a principal funcionalidade que distingue o Application Insights do New Relic. O utilizador pode criar eventos que não são registados automaticamente pelo *browser*, através da introdução de pedaços de código, no local onde se deseja disparar o evento.

Pode ser utilizado, por exemplo, para registar o número de vezes que é feito o render de uma *partial view*[13] ou quantas vezes um cliente utiliza determinada funcionalidade.

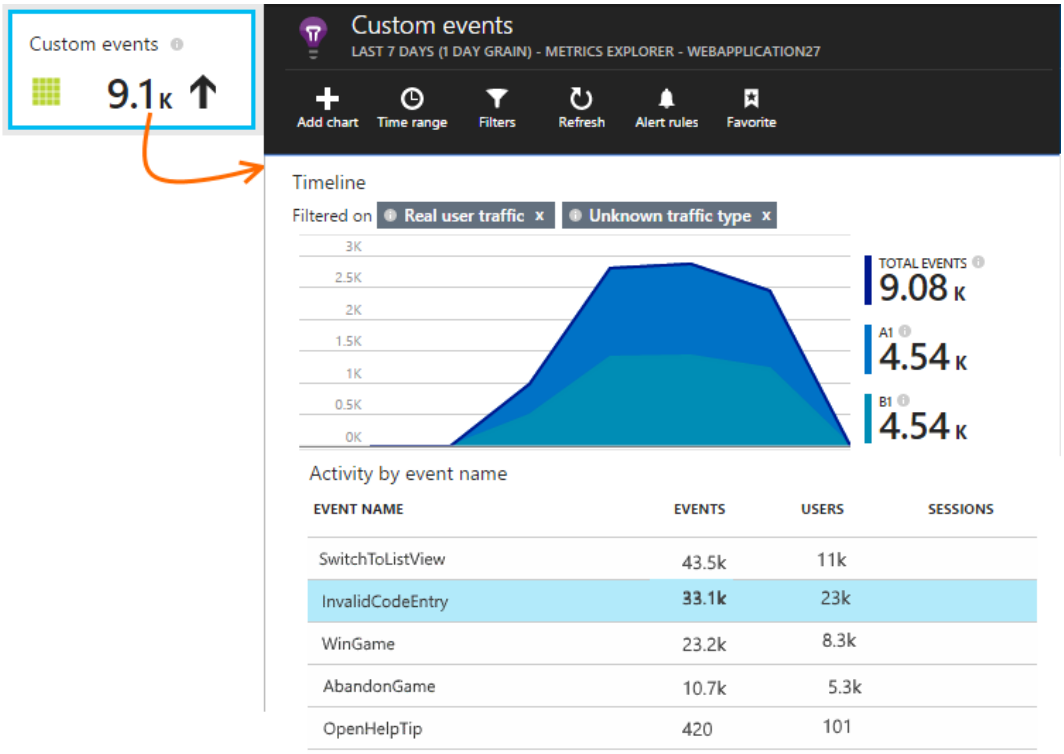

Figura 2.12: Application Insights for Javascript.[Wil15a]

Estes eventos encontram-se visíveis, agregados no explorador de métricas, ou no *"Diagnostic Search"* para serem consultados e explorados individualmente.

Exportar

Por fim existe também a opção de exportar dados em formato JSON para serem posteriormente manipulados, caso seja desejado executar uma análise diferente da realizada por esta API [Wil15c].

Este serviço dispõe ainda da funcionalidade de alterar o tipo de gráficos e manipular de que forma a informação destes é apresentada. Em suma, o Application Insights, apesar de ter uma licença de utilização, parece bastante mais completo que o New Relic.

[13] Designação atribuída a uma página que pode ser desenhada dentro de outra. Uma *partial view* pode ser utilizada varias vezes, reduzindo a repetição de código.

2.1.5 Speed Index

Anteriormente foram apresentadas duas API's que em conjunto têm um papel dominante na avaliação da performance de uma página, mas infelizmente não reportam exatamente a experiência do utilizador, na medida em que uma página pode carregar em 5 segundos, mas o conteúdo visível para o utilizador ser apresentado em menos tempo.

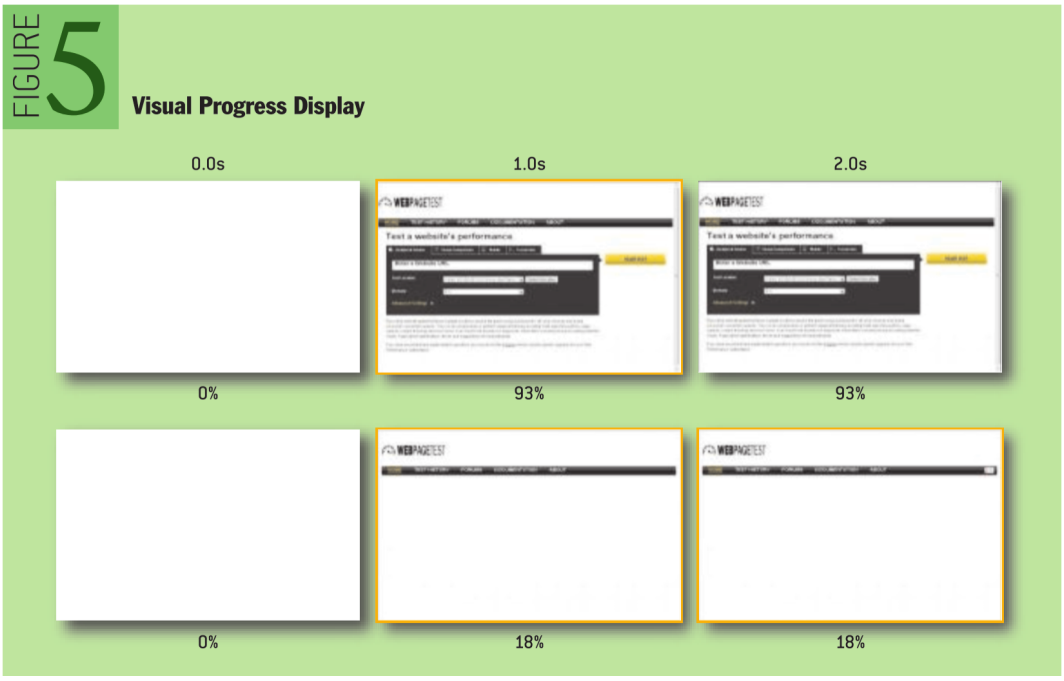

Figura 2.13: Exemplo de uma análise utilizando a métrica *Speed Index* [Mee13].

A métrica *Speed Index*, consiste em analisar a percentagem de pixeis visíveis para o utilizador e assim reportar corretamente e com precisão o *delay* resultante para o utilizador. Esta técnica foi testada e resulta como um melhoramento às estudadas anteriormente. É de salientar que, até ao momento é apenas possível a sua análise em laboratório [Mee13].

2.2 Gestão de Memória e Processador

Quando se fala de *web profiling*, para além do tempo que a página demora a carregar temos também a questão da utilização de memória e processador por parte da mesma.

Um dos objetivos mais usuais, na recolha de informação acerca da memória e processador em utilização, é o ajuste automático por parte da aplicação, como por exemplo, a redução da qualidade de vídeo e imagem, consoante a percentagem de utilização e também o tipo de processador, para assim, fornecer uma melhor experiência para o utilizador. O objetivo nesta dissertação, é utilizar estas métricas para identificar de que forma, estas podem influenciar a utilização da aplicação de modo negativo e qual o possível caminho para solucionar o problema.

2.2.1 Gestão de memoria

O processo de gestão de memória consiste em três momentos: é feita a alocação necessária de memória; essa memória é utilizada para ler/escrever; por fim, libertada quando não é mais necessária. Ao contrário de outras linguagens, que providenciam funções como, *malloc()*, que permitem fazer essa gestão, o Javascript, atribui a memória no momento de declaração da variável e liberta essa memória automaticamente, quando não é mais necessária. Este processo automático de gestão de memória, é designado de *garbage collector*.

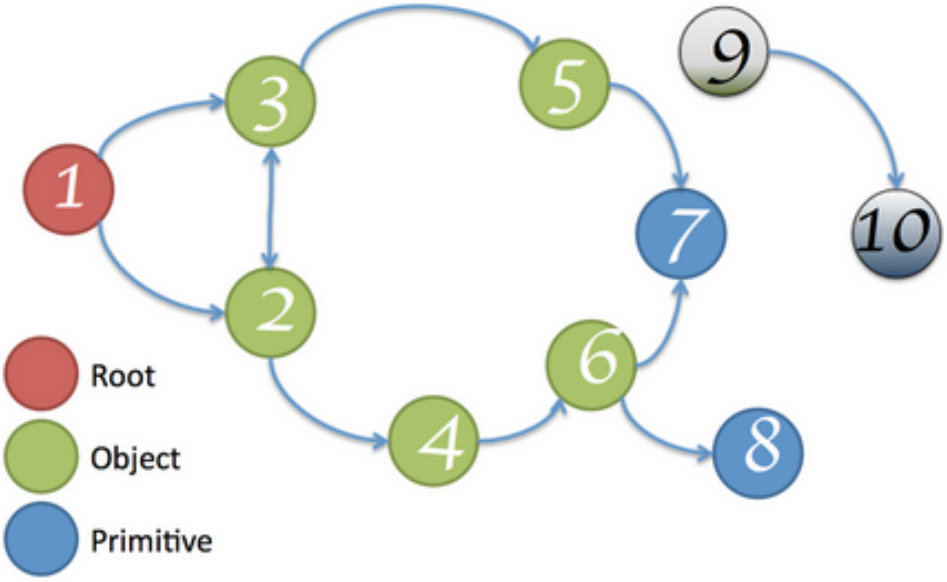

Figura 2.14: Gráfico de memória com objectos prontos a serem eliminados pelo *garbage collector*. [Devb]

O *garbage collector* é responsável por fazer a gestão da memória, ou seja, monitoriza todos os objetos da aplicação que estão a ser ou podem ser utilizados, bem como todos os que já não estão em uso ou que não podem ser alcançados pelo código. Na Figura 2.14, está visível uma situação em que o *garbage collector* pode eliminar os objetos 9 e 10, na medida em que estes não são alcançáveis.

Atualmente, por questões de segurança, a informação relativa à utilização de memória por parte de uma aplicação, não é fornecida ao utilizador para análise dinâmica e generalizada. Existe no entanto, a possibilidade dessa análise, através da criação de extensões, no caso de análise automática, para o tipo de *browser* em uso, ou então, a utilização dos *developer tools* já embutidos no próprio.

Todas estas restrições, fazem com que esta área de *web profiling* seja demasiado especifica, pois é necessário escolher em que *browser* se deseja executar o estudo.

2.2.1.1 Memory Leak

Um *memory leak* é uma perda progressiva de memória por parte de uma aplicação, que falha repetidamente em retornar a espaço que requisitou para utilização temporária. No caso do Javascript, o significado é o mesmo, pelo que o problema surge no momento de libertação da mesma, ou seja, quando o *garbage collector* não consegue determinar corretamente, se um objeto já não se encontra em uso. Isto provoca incapacidade de reaproveitar a memória alocada para este. Estes problemas são comuns devido a erros de código que acabam por prejudicar a performance da aplicação [Net15a][Devb].

Este dilema era ignorado anteriormente pelo facto de a memória ser refrescada, cada vez que se navegava para uma página diferente, mesmo no caso de ser no interior da mesma aplicação. No entanto, esta questão foi alvo de interesse, a partir do momento em que surgiu a Web 2.0 [GG14b] com a utilização da tecnologia Ajax, o que tornou as aplicações mais dinâmicas e possibilitou a navegação, sem a necessidade de fazer o refrescamento à página [RGEV11].

2.3 Gestão de Processador

Ao contrário de outras linguagens de programação, onde é possível obter informação sobre a utilização do processador, no Javascript, sendo esta linguagem destinada à programação web e consequentemente acessível a atores externos, o acesso a este poderia ser extremamente perigoso a nível de segurança.

No entanto, a *Intel*[14] oferece a possibilidade de recolher essa informação [(In10] e assim, através da mesma, os *developers* da aplicação podem fornecer uma melhor experiência de utilização para o cliente. A questao da segurança pode ser contornada utilizando diferentes técnicas de proteção, como é o exemplo da ofuscação de código [XZZ12].

Browser	Sistema Operativo	Comentários
Firefox 3.5.0+	Windows (Xp,Vista,7)	Todas as funcionalidades são suportadas.
	Linux, Mac OSx	Não existe suporte de momento.
Chrome 4.0.0+	Windows (Xp,Vista,7)	Todas as funcionalidades são suportadas.
	Linux, Mac OSx	Não existe suporte de momento.
Safari 4.0.0+	Windows (Xp,Vista,7)	Todas as funcionalidades são suportadas.
	Linux, Mac OSx	Não existe suporte de momento.
Opera 10.0+	Windows (Xp,Vista,7)	Existe problemas com algumas funcionalidades.
	Linux, Mac OSx	Não existe suporte de momento.
Internet Explorer 8.0+	Windows (Xp,Vista,7)	Não existe suporte de momento.
	Linux, Mac OSx	Não existe suporte de momento.

Tabela 2.1: Compatibilidade de Browser e Sistema Operativo

Como é possível verificar, o problema da compatibilidade na Tabela 2.1, prejudica mais uma vez a possibilidade de criar uma aplicação generalizada, que permita alcançar o objetivo desejado [(In10].

[14]Empresa Americana, multinacional e líder de mercado na criação de microchips como é o caso dos processadores.

A informação acerca da utilização do processador pode, no entanto também ser obtida da mesma forma que a utilização de memória, através de extensões ou utilizando o *developer tools*.

2.4 Código Javascript

Atualmente, o Javascript é uma das linguagem mais utilizadas em todo o mundo tecnológico, principalmente em desenvolvimento web, tanto para *Client-Side* como *Server-Side*. É usual ao pesquisar por Javascript nos depararmos com o termo *ECMAScript*[15]. Este termo corresponde a um padrão para linguagens de script e as funções principais de Javascript são baseadas neste padrão [Boe12][RLBV10].

Neste capítulo vai ser aprofundado o conhecimento sobre a análise de código Javascript, através do *parsing* do mesmo, para uma árvore sintática que respeita os padrões do *ECMAScript* e posteriormente o seu respetivo processamento e avaliação.

A análise da árvore sintática, permite identificar erros ou ineficiências no código da aplicação, com o intuito de fornecer ao utilizador sugestões para corrigir esses problemas e assim aumentar o desempenho.

2.4.1 Análise Sintática

Parsing ou análise sintática, consiste em explorar uma sequência de código, para determinar a sua estrutura segundo uma determinada gramática. Esta análise transforma o código de entrada, numa estrutura de dados que geralmente é uma árvore sintática, onde estão representados os nós, bem como a relação entre estes [KNB+05].

Este processo é também utilizado nos compiladores juntamente com a análise lexical e semântica, como por exemplo, no caso de linguagens de programação como, *C++*, *Java*, entre outras. Após o desenvolvimento do código, é executado o *parsing* pelo compilador do mesmo. É após este processo que são identificados possíveis erros e problemas que impedem o código de ser corretamente executado.

O que torna o Javascript uma linguagem bastante complexa de desenvolver, é precisamente por não ser realizado todo processo exercido por um compilador. O código é executado pelo *browser* independentemente de existirem inconsistências na sua construção.

2.4.2 Esprima Parser

A linguagem Javascript cresceu imenso em termos de utilização e popularidade. Anteriormente era extremamente difícil fazer uma análise sintática deste código, devido à sua natureza dinâmica, mas essa, ficou bastante mais facilitada com o *Esprima*.

Esprima é um *parser* de código Javascript criado e mantido por Ariya Hidayat e é compatível com os *browsers* mais utilizados, bem como, outras plataformas baseadas em *ECMAScript* como é

[15] Padrão de linguagens script implementado por linguagens como é exemplo: JScript ActionScript e Javascript.

o caso de *Rhino*[16] e *Node.js*[17]. Outra vantagem deste *parser*, é o seu código fonte ser relativamente fácil de compreender e alterar, no caso de haver necessidades específicas na tradução do código Javascript para a árvore sintática [Hid][Boe12].

O aparecimento desta e de outras API's, deve-se à exposição do *parser*, usado pelo motor *SpiderMonkey* do Mozzila Firefox como uma API Javascript, proporcionando uma melhoria exponencial na análise e manipulação desta linguagem. A documentação da API pode ser encontrada no site oficial da *Parser API* [Net].

A finalidade deste *parser* é transformar código Javascript em uma AST (árvore sintática abstrata) em formato JSON compatível com a Mozzila Parser API. Esta representação confere a possibilidade do código fonte ser analisado programaticamente.

A utilização é bastante simples, sendo apenas necessário incluir o *script* da API e de seguida, fazer o respetivo *parsing* do código desejado, como é explicado a seguir:

```
1 <script src="esprima.js"></script>
2 var AST = esprima.parse(code, {tokens:true});
3 tokens = AST.tokens;
```

Listing 2.3: Código Javascript necessário para utilizar o Esprima.

Como é visível na Listing 2.3[18], a função *"parse"* pode ser invocada com um conjunto de opções, que quando o seu valor é igual a *"true"*, fornecem informação ou parâmetros extra aos componentes da árvore sintática:

loc - os nós da AST apresentam informação sobre a linha e a coluna da sua localização.

range - os nós têm informação sobre o índice e tamanho.

tokens - com esta opção a *"true"*, é exposto um array com todos os tokens encontrados.

comment - fornece um array extra contendo todos os comentários.

tolerant - esta opção, permite à função ser tolerante e tentar continuar o *parsing* do código, mesmo quando um erro é encontrado. No fim, é retornado um array com os erros.

Segue-se, a apresentação de um excerto de código Javascript e a árvore sintática correspondente, gerada através do *Esprima*, executando a função com o parâmetro *tokens* igual a *"true"*.

[16] Plataforma *open-source* de Javascript escrita em Java.

[17] Plataforma de construída no motor de Javascript do Google Chrome para criar aplicações de rede rápidas e escaláveis.

[18] http://esprima.org/doc/index.html

```
var a = 1;
var b = 1;
function add2Numbers(a, b) {
    return a + b;
}
var c = add2Numbers(a,b);
```

Listing 2.4: Excerto de código Javascript usado para obter a AST.

Figura 2.15: Javascript AST obitda do excerto anterior.

Neste excerto de código, foram obtidos 23 nós visíveis na árvore sintática da Figura 2.15 e 34 tokens a seguir evidenciados, em que cada linha de tokens, representa a linha respetiva de código Javascript.

```
Keyword(var) Identifier(a) Punctuator(=) Numeric(1) Punctuator(;)
Keyword(var) Identifier(b) Punctuator(=) Numeric(1) Punctuator(;)
Keyword(function) Identifier(add2Numbers) Punctuator(() Identifier(a) Punctuator(,)
     Identifier(b) Punctuator()) Punctuator({)
Keyword(return) Identifier(a) Punctuator(+) Identifier(b) Punctuator(;)
Punctuator(})
Keyword(var) Identifier(c) Punctuator(=) Identifier(add2Numbers) Punctuator(()
    Identifier(a) Punctuator(,) Identifier(b)Punctuator())Punctuator(;)
```

Segundo o padrão *ECMAScript*, um token é um elemento de entrada que não seja um espaço em branco ou um comentário, de todos os padrões possíveis para a gramática *ECMAScript* e podem ser de cinco tipos diferentes: Identifier, Punctuator, Numeric, String ou Keyword [int11][Boe12].

Como um dos principais objetivos, é detetar problemas no código e aumentar a *performance*, esta API tem também o objetivo de não produzir aumentos exagerados no *parsing* do código e atualmente é o *parser* que executa a sua função com os melhores tempos.

Após obter a AST, é factível a análise sintática para produzir resultados como a procura por variáveis não declaradas, expressões regulares, número de vezes que uma função é executada,

entre muitas outras.

Apesar de não serem analisadas as diferentes possibilidades de utilização deste *parser*, como este é *open-source*, existe uma comunidade empenhada em criar módulos que o complementem, para obter o máximo proveito do mesmo e como todas as outras ferramentas, para aumentar a performance das aplicações.

2.4.3 Boas práticas

Existem diversas técnicas já identificadas, que melhoram, e em muitos casos consideravelmente o desempenho das aplicações web [Sou08][Nic13][Nag13].

Diminuir o que é enviado para o browser

Quando um utilizador acede a uma página web, os diversos recursos são carregados, como por exemplo imagens, ficheiros Javascript, CSS, entre outros. Otimizar este ponto, deve ser a principal prioridade para melhorar o tempo de carregamento da página. É possível obter estes resultados removendo todos os recursos que sejam desnecessários. Uma outra opção, é a compressão de dados. Alguns dos *browsers* mais recentes já oferecem essa possibilidade, o que reduz bastante a quantidade de dados que são transferidos para o lado do cliente.

Minificar recursos

A minificação de um recurso é o processo de remoção de espaços, linhas em branco e comentários que não são necessários e em que a sua remoção não afeta o funcionamento correto do mesmo. É também comum fazer a combinação de diversos recursos para minimizar o número de chamadas HTTP. Esta é uma técnica bastante comum e utilizada para diminuir consideravelmente o tempo de carregamento de uma aplicação web.

Utilizar a cache do Browser

Quando um utilizador visita uma página pela primeira vez, é carregado o HTML em conjunto com todos os recursos necessários ao seu correto funcionamento. De seguida todos estes componentes são armazenados na *cache* do *browser*, o que faz com que nas próximas visitas não seja necessário executar novamente a chamadas HTTP para o servidor e o carregamento dos recursos é mais rápido, porque estes estão retidos localmente. Tudo isto faz com que o tempo de carregamento da página seja notoriamente mais rápido após a primeira visita. Pelas razões explicadas anteriormente, se a performance da aplicação é uma preocupação, a utilização da *cache* é imprescindível.

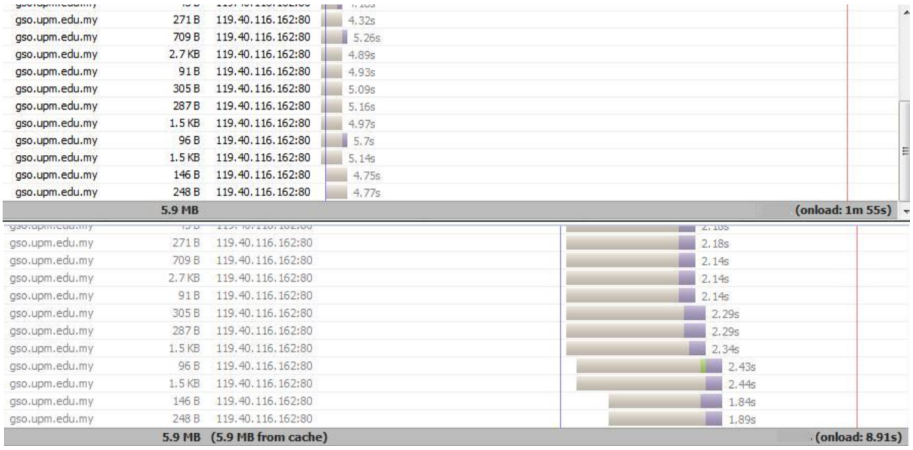

Figura 2.16: Diferença entre o primeiro e segundo carregamento utilizando o *browser* Mozzila Firefox [YIHU13].

Otimizar imagens

Quando se fala de imagens, temos de ter em conta diversos fatores, sendo eles: o tamanho, formato e local de armazenamento. No caso do formato, a utilização de imagens ".jpg", ".png"ou ".gif"em alguns casos é recomendado. O tamanho é a questão mais pertinente quando falamos de otimizar uma imagem, visto que carregar uma imagem de dimensão grande é bastante demorado. Deve-se reduzir o tamanho para o do seu *container*, para que não seja criado um aumento desnecessário no seu carregamento. Também é importante ter em conta se é necessário através de CSS por exemplo, alterar a imagem consoante o tamanho do ecrã, para não existir, uma imagem destinada a um ecrã de 1920x1080 a ser carregada num dispositivo móvel.

Diminuir o número de redirects

Um *redirect* resulta num aumento de chamadas HTTP e consequentemente do tempo de carregamento da página. Por esta razão deve-se evitar ao máximo a sua utilização desnecessária.

Monitorizar a aplicação

Esta é provavelmente uma das práticas mais importantes a ter em conta, senão a mais importante. Todas as práticas apresentadas anteriormente, devem ser continuamente aplicadas para atingir os melhores resultados possíveis.

É também importante ao monitorizar uma aplicação, ter em atenção qual são os patamares desejados, para saber até que ponto existe um problema de performance e em caso afirmativo, saber até que ponto se deve otimizar.

Esta dissertação tem como foco principal esta prática, explorando técnicas que são utilizadas atualmente, mas também possibilidades mais inovadoras.

2.5 Resumo ou Conclusões

A questão da performance web deixou de ser secundária e passou a ser uma preocupação no dia-a-dia de empresas e *developers*. Tendo em conta as tecnologias analisadas neste capítulo, apesar de ser notório estarmos ainda no inicio do *web profiling*, já existem algumas soluções que permitem melhorar a qualidade e velocidade das aplicações, sem prejudicar a questão do *User Friendly* introduzido pela *Web 2.0*.

Capítulo 3

Descrição e Projeto

A constante expansão da empresa Glintt HS e o aumento consequente do número de clientes e aplicações desenvolvidas para os mesmos, faz com que a necessidade de os satisfazer seja uma das principais prioridades. Esta satisfação é necessária tanto ao nível da qualidade e velocidade do software desenvolvido, como da sua capacidade em responder aos problemas e falhas que ocorrem nas suas aplicações. Atualmente um problema apenas é detetado e solucionado após o cliente exercer o contacto com a empresa, referindo que existiu um falha em determinada aplicação. O JS Performance Certifier vem colmatar esta questão, na medida em que regista toda a interação do cliente com a aplicação, o que permite num só local verificar onde existem problemas e para alguns casos como os solucionar. Este sistema permite identificar diversas ineficiências para posteriormente oferecer uma experiência de utilização mais rápida e eficiente.

A solução proposta descrita nesta dissertação, foi a criação e implementação de um sistema que englobasse os seguintes objetivos específicos:

- Análise e estudo de ineficiências de performance aplicacional e criação de dicionários de correção/melhoria.
- Desenvolvimento de solução de deteção de ineficiências de performance aplicacionais e ferramentas de monitorização e sugestões de correção.
- Elaboração de relatório de ineficiências e de certificação de performance aplicacional de um caso prático.

Para criar esta solução foi necessário desenvolver os seguintes componentes:

Biblioteca Javascript responsável por obter toda a informação relativa à interação entre o utilizador e a aplicação web.

Serviço Windows necessário para interagir com a biblioteca e a aplicação Windows. O serviço executa toda a gestão necessária, para que a aplicação Windows consiga monitorizar os processos corretamente.

Aplicação Windows recolhe informação sobre a utilização de memória e processador relativa a um processo, que está responsável pelo separador do *browser*, onde a aplicação é ser executada.

Serviço REST local para onde são enviados os dados recolhidos nos clientes. Esta informação é armazenada na base de dados.

DashBoard neste componente, é possível visualizar a informação armazenada pelos componentes anteriores.

Este capítulo está dividido em três secções. Na primeira secção são expostos os requisitos funcionais e não funcionais da solução, assim como os atores do sistema, narrativas, casos de utilização e por fim um *project charter*. Na segunda secção é feita a apresentação da arquitetura proposta, é demonstrada uma visão geral de todo o sistema e de seguida, são apresentados os ecrãs de navegação existentes e a arquitetura física de todos os componentes.

3.1 Requisitos do Sistema

Nesta secção estão enumerados todos os objetivos que o produto final teve de satisfazer de acordo com as necessidades e requisitos da Glintt HS. Para completar é também apresentado quais os atores que interagem com o sistema desenvolvido bem como as respetivas narrativas e casos de utilização.

3.1.1 Requisitos Funcionais

Os requisitos funcionais são objetivos que os vários componentes do sistema têm de cumprir. Devido ao facto de a solução final ser extremamente extensa e complexa, é feita uma apresentação dos requisitos segmentada por componente.

Biblioteca Javascript (JSPCRecorder)

Esta biblioteca necessita de ser incluída nas aplicações web que se deseja monitorizar. Todo o processo é inicializado a partir deste componente.

ID	Nome	Descrição	Prioridade
R01.01	Iniciar sessão	A biblioteca deve inicializar uma sessão por cada separador do *browser*. Cada sessão fica identificada através da criação de um ID único.	Essencial.
R01.02	Obter eventos	Deve receber informação por cada interação do utilizador com a aplicação.	Essencial
R01.03	Obter funções	A biblioteca deve registar todas as funções que são executadas durante o tempo de vida da aplicação web.	Importante
R01.04	Obter erros	A biblioteca tem de registar a informação quando um erro ocorre.	Essencial
R01.05	Obter chamadas ajax	A biblioteca tem de registar as chamadas Ajax e a sua respetiva informação.	Essencial
R01.06	Enviar informação	A biblioteca tem enviar os dados recolhidos para o serviço web que de seguida executa o seu devido tratamento e registo na base de dados.	Essencial
R01.07	Carregamentos de página	A biblioteca deve lidar com o carregamento e refrescamento da aplicação web, sendo necessário neste ultimo caso enviar os dados que ainda não foram registados, para o serviço REST.	Desejável
R01.08	Comunicar com o Windows Service	A biblioteca deve realizar a comunicação com o windows service para informar que uma nova aplicação foi inicializada.	Essencial

Tabela 3.1: Requisitos da biblioteca javascript.

Windows Service (JSPCService)

O serviço do Windows é a ponte de ligação entre as páginas web e a aplicação executada no ambiente do utilizador.

ID	Nome	Descrição	Prioridade
R02.01	Receber comunicações das aplicações web	O serviço deve receber a comunicação no momento de inicialização de uma página web.	Essencial
R02.02	Executar aplicação no ambiente do utilizador	O serviço tem de ter a capacidade de lançar uma aplicação (JSPCAnalyzer) no ambiente de trabalho do utilizador, no momento do *login*.	Essencial
R02.03	Gerir diversos utilizadores	O serviço deve ter a destreza de gerir varias sessões no mesmo computador, nomeadamente para o caso de sessões remotas.	Essencial
R02.04	Comunicar com as aplicações executadas	O serviço deve enviar e receber informação de todas as aplicações executadas nas diferentes áreas de trabalho.	Essencial
R02.05	Gerir refrescamento de aplicações web	O serviço deve ter a capacidade de perceber quando uma página é refrescada ou terminada, para informar a aplicação responsável pela sua monitorização.	Desejável
R02.06	Gerir processos	O serviço tem de monitorizar os processos do Windows, para saber quando um novo *browser* é inicializado. De seguida é comunicado apenas à aplicação no ambiente em particular.	Essencial

Tabela 3.2: Requisitos do serviço do Windows.

Windows Application (JSPCAnalyzer)

Esta aplicação é responsável por acompanhar todos os *browsers* e separadores. Em caso de comunicação de uma nova aplicação web que tenha incluída a biblioteca JSPCRecorder, é feita a monitorização do processo responsável pela mesma.

ID	Nome	Descrição	Prioridade
R03.01	Receber comunicações do serviço Windows	A aplicação tem de comunicar com o serviço presente na mesma máquina	Essencial
R03.02	Acompanhar *browser*	A aplicação deve acompanhar um *browser*, caso o serviço comunique que foi inicializado no seu ambiente.	Essencial
R03.03	Detetar novos separadores	Para cada *browser* monitorizado, deve registar quando um novo separador é criado.	Essencial
R03.04	Detetar separadores eliminados	Para cada *browser* monitorizado, deve registar todos os separadores que são fechados.	Essencial
R03.05	Iniciar a monitorização	Quando uma nova aplicação web com a biblioteca Javascript é inicializada, a aplicação deve monitorizar o processo do separador em questão.	Essencial
R03.06	Parar a monitorização	A aplicação deve parar a monitorização de um processo, no momento de comunicação do serviço com a informação que o separador foi eliminado.	Importante
R03.07	Registar dados	A aplicação deve guardar na base de dados, todos os registos de utilização de processador e memória de um processo, periodicamente.	Importante

Tabela 3.3: Requisitos da aplicação Windows.

Servidor (JSPCRest)

Servidor com um serviço que implementa a arquitetura REST. É responsável por receber comunicações da biblioteca Javascript e Aplicação Windows através de pedidos HTTP, registando de seguida toda essa informação na base de dados.

ID	Nome	Descrição	Prioridade
R04.01	Iniciar sessão	No momento da comunicação por parte da aplicação web, o servidor deve inicializar uma nova sessão e registá-la na base de dados.	Essencial
R04.02	Registar dados	Aquando da comunicação por parte de um dos componentes, o servidor deve manipular e registar essa informação na base de dados, na sessão indicada.	Essencial
R04.03	Fornecer dados	O servidor deve abastar a *Dashboard* com os dados das sessões registadas previamente.	Essencial
R04.04	Atualizar sumário	A API deve atualizar o sumário da sessão após qualquer comunicação.	Desejável

Tabela 3.4: Requisitos do serviço REST.

Aplicação Web (DashBoard)

Esta aplicação recolhe os dados armazenados na base de dados. Oferece uma visualização detalhada e de fácil uso de toda a interação entre clientes e páginas web.

ID	Nome	Descrição	Prioridade
R05.01	Listar sessões	A aplicação deve permitir listar sessões registadas.	Essencial
R05.02	Filtrar sessões	A aplicação deve permitir filtrar as sessões por data e url.	Essencial
R05.03	Páginas com erros	A aplicação deve fornecer a opção de verificar quais as páginas com erros.	Essencial
R05.04	Sumário Ajax	A aplicação deve permitir analisar informação sobre todas as chamadas Ajax.	Essencial
R05.05	Páginas visitadas	A aplicação deve permitir verificar quais as páginas mais visitadas.	Desejável
R05.06	Análise de funções	A aplicação deve permitir analisar código Javascript. Tanto funções executadas nas sessões como ficheiros completos.	Essencial
R05.07	Analisar sessão	Deve ser possível selecionar uma sessão e analisa-la detalhadamente.	Essencial

Tabela 3.5: Requisitos da aplicação web.

3.1.2 Requisitos não funcionais

Os requisitos não funcionais relacionam-se com o uso da aplicação tendo em conta vários critérios, como por exemplo, disponibilidade, desempenho e versatilidade. Os requisitos a seguir expostos são os considerados mais importantes:

Biblioteca Javascript (JSPCRecorder)

RNF.01 A biblioteca não deve proporcionar uma diminuição na qualidade ou velocidade das aplicações.

RNF.02 A biblioteca deverá registar 90% da interação com a aplicação web.

Windows Service (JSPCService)

RNF.03 O serviço deve ser inicializado com o Sistema Operativo.

RNF.04 O serviço deverá operar com qualquer Sistema operativo.

RNF.05 O serviço deve estar preparado para múltiplas comunicações.

RNF.06 O serviço necessita ser desenvolvido em vista a utilizar .Net Framework 3.5.

RNF.07 O serviço deve estar preparado para lidar com 254 sessões diferentes.

Windows Application (JSPCAnalyzer)

RNF.08 A aplicação deverá operar com qualquer Sistema Operativo.

RNF.09 A aplicação deverá operar com Google Chrome 45+, Microsoft Edge, Microsoft Internet Explorer 8+.

RNF.10 A aplicação necessita ser desenvolvido em vista a utilizar .Net Framework 3.5.

Web Service (JSPCREST)

RNF.11 O serviço necessita ser desenvolvido para .Net Framework 4.0.

RNF.12 A aplicação deverá operar com Google Chrome 45+, Microsoft Edge, Microsoft Internet Explorer 8+.

Aplicacao Web (JSPCDashBoard)

RNF.13 O aplicação necessita ser desenvolvido de modo a utilizar .Net Framework 4.0.

3.1.3 Atores

Os possíveis atores presentes nesta solução estão divididos em dois tipos e serão enunciados na tabela seguinte.

Nome	**Permissões**
Cliente - Glintt HS	Utilizador que interage com as aplicações da empresa. Muito provavelmente não se apercebe do que está a acontecer, porque acontece tudo em *background*.
Engenheiro de Software - Glintt HS	Utilizador que pode analisar a informação na DashBoard.

Tabela 3.6: Atores do sistema.

3.1.4 Narrativas de Utilização

Estão consideradas as seguintes narrativas de utilização para a solução proposta, com o esforço estimado usando a escala de Fibonacci e prioridade.

ID	Nome	Descrição	Prioridade	Esforço
US01	Listar sessões	Como ES quero listar e filtrar sessões.	Essencial	5
US02	Analisar sessão	Como ES quero analisar uma sessão detalhadamente.	Essencial	8
US03	Páginas mais visitadas	Como ES quero saber as páginas mais visitadas em determinado espaço de tempo.	Desejável	3
US04	Verificar ocorrência de erros	Como ES quero saber os erros que ocorreram, durante a utilização das aplicações Glintt HS em determinado espaço de tempo.	Desejável	3
US05	Chamadas Ajax	Como ES quero obter informação acerca de todas as chamadas Ajax, de todas as sessões, em um determinado espaço de tempo.	Desejável	3
US06	Análise de funções	Como ES quero saber que funções são executadas em uma sessão.	Importante	8
US07	Sugestões de correção	Como ES para cada função quero obter sugestões de correção ou otimização caso se apliquem.	Importante	3
US08	Analisar ficheiros	Como ES quero ter a oportunidade de verificar se um ficheiro Javascript, contém qualquer tipo de ineficiência e ao mesmo tempo ser informado de possíveis soluções.	Importante	3

Tabela 3.7: Narrativas de utilização.

3.1.5 Casos de Utilização

O modelo de casos de utilização permite relacionar os atores existentes com as ações que estes podem realizar. É uma representação externa e de alto nível do sistema que representa as funcionalidades do produto acessíveis ao utilizador final.

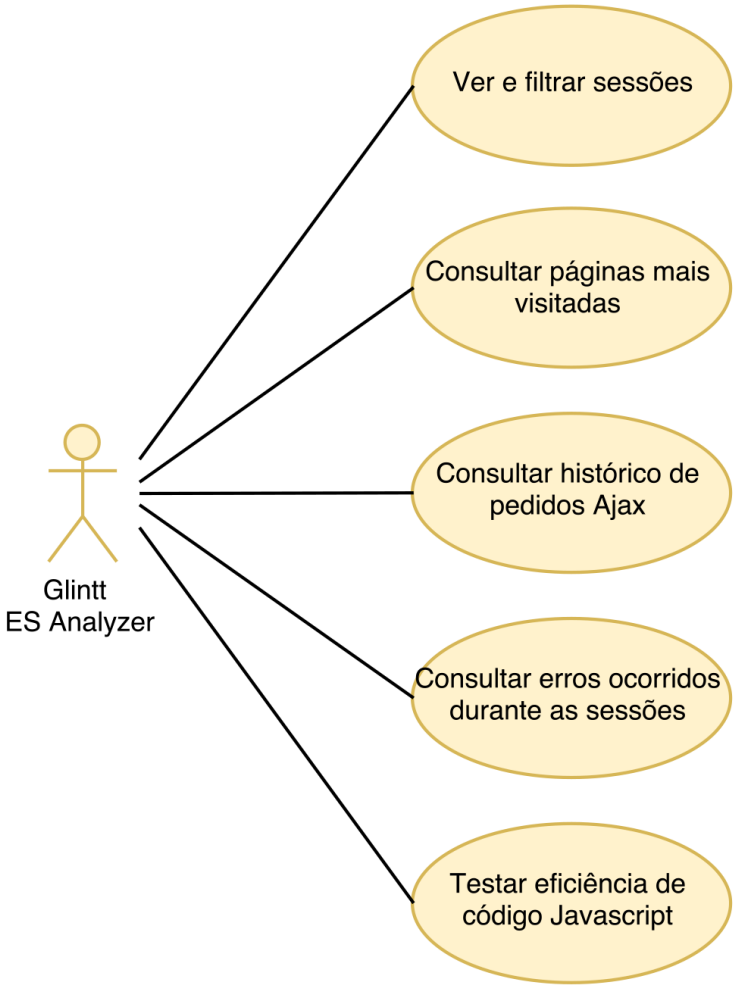

Figura 3.1: Diagrama de casos de utilização do menu.

Identificador	UC01
Nome	Ver e filtrar sessões
Descrição Sumária	Permite ao utilizador analisar as sessões que foram registadas durante um determinado espaço temporal. Para maior especificação existe também a possibilidade de pesquisa por url.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Ter acesso à rede interna da Glintt HS.
Pós-Condições	O utilizador visualizará todas as sessões que respeitam as condições de filtragem.
Procedimento	1. Entrar na aplicação 2. Clicar no botão de Menu. 3. Clicar em Overview.

Identificador	UC02
Nome	Consultar páginas mais visitadas
Descrição Sumária	Oferece ao utilizador uma visão geral do número total de visitas às aplicações web. Esta visão está divida pelo url da página.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Ter acesso à rede interna da Glintt HS.
Pós-Condições	O utilizador visualizará um gráfico de barras, em que cada uma representa o total de visitas a uma página web, num determinado espaço de tempo.
Procedimento	1. Clicar no botão de Menu. 2. Clicar em Visited Pages.

Identificador	UC03
Nome	Consultar histórico de pedidos Ajax.
Descrição Sumária	Permite ao utilizador ver os pedidos Ajax efetuados numa aplicação ou mais aplicações, num determinado espaço de tempo.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Ter acesso à rede interna da Glintt HS.
Pós-Condições	O utilizado passará a ter acesso à visualização de todos os pedidos Ajax, num espaço de tempo específico.
Procedimento	1. Clicar no botão de Menu. 2. Clicar em Ajax Summary.

Identificador	UC04
Nome	Consultar erros ocorridos durante as sessões.
Descrição Sumária	Permite ao utilizador uma visão detalhada de todos os erros que ocorreram durante um determinado espaço de tempo, nas suas aplicações.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Ter acesso à rede interna da Glintt HS.
Pós-Condições	O utilizado visualizará um gráfico de todos os erros que ocorreram no espaço de tempo escolhido.
Procedimento	1. Clicar no botão de Menu. 2. Clicar em Aplication Errors.

Identificador	UC05
Nome	Testar eficiência de código Javascript.
Descrição Sumária	Permite ao utilizador aceder à área de teste e certificação de código Javascript.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Ter acesso à rede interna da Glintt HS.
Pós-Condições	O utilizador estará presente na área de teste de código Javascript.
Procedimento	1. Clicar no botão de Menu. 2. Clicar em Javascript Certification.

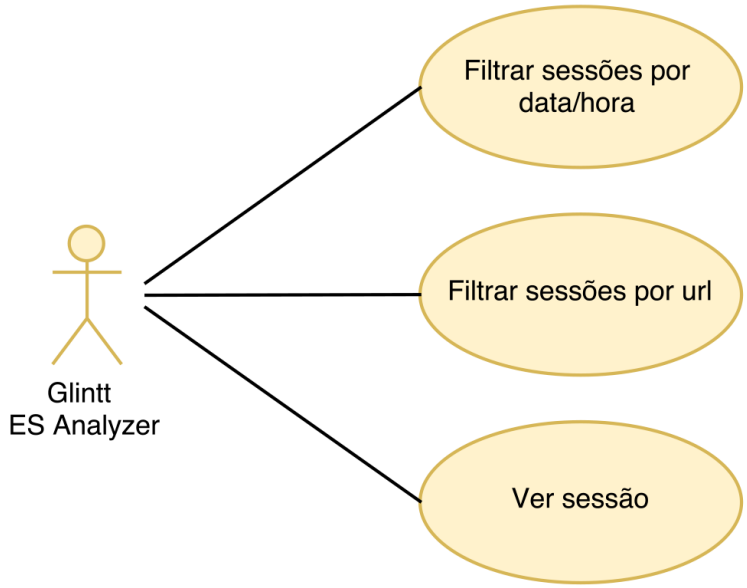

Figura 3.2: Diagrama de casos de utilização da área de sessões.

Identificador	UC06
Nome	Filtrar sessões por data/hora.
Descrição Sumária	Permite ao utilizador definir um espaço de tempo para filtrar as sessões. Opções predefinidas: hoje, ontem, 7 dias, 30 dias, mês passado ou possibilidade de definir uma data e hora customizadas.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área da lista de sessões.
Pós-Condições	O utilizador tem acesso às sessões desejadas.
Procedimento	1. Clicar no calendário. 2. Selecionar uma opção predefinida ou se desejável, definir manualmente.

Identificador	UC07
Nome	Filtrar sessões por URL.
Descrição Sumária	Permite ao utilizador visualizar as sessões onde o cliente acedeu a uma determinada página.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área da lista de sessões.
Pós-Condições	O utilizador visualizará as sessões que contém páginas visitadas com o URL introduzido.
Procedimento	1. Clicar na caixa de pesquisa. 2. Introduzir o URL da página desejada.

Identificador	UC08
Nome	Ver sessão.
Descrição Sumária	Permite ao utilizador visualizar num gráfico, a *timeline* da sessão com todos os detalhes da mesma.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área da lista de sessões.
Pós-Condições	O utilizador vê a sessão selecionada.
Procedimento	1. Clicar no botão Inspect Session de uma sessão.

Figura 3.3: Diagrama de casos de utilização da área de inspeção de uma sessão.

Identificador	UC09
Nome	Ajustar limites do gráfico de memória.
Descrição Sumária	Permite ao utilizador ajustar os limites da linha de memória, presente na *timeline* da sessão, desde 100MB até 2000MB.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de inspeção de uma sessão.
Pós-Condições	O utilizador visualiza em tempo real a alteração dos limites do gráfico.
Procedimento	1. Arrastar a barra de Memory Ajustment.

Identificador	UC10
Nome	Selecionar range de tempo na pré-visualização da timeline.
Descrição Sumária	Permite ao utilizador selecionar uma porção da *timeline* que deseja consultar.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de inspeção de uma sessão.
Pós-Condições	O utilizador vê na *timeline* apenas o espaço de tempo que selecionou.
Procedimento	1. Clicar na pré-visualização, no momento inicial desejado. 2. Arrastar até ao momento final.

Identificador	UC11
Nome	Arrastar*timeline* para focar os itens desejados.
Descrição Sumária	Permite ao utilizador arrastar a *timeline*, mesmo que ultrapasse o limite selecionado na pré-visualização.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de inspeção de uma sessão.
Pós-Condições	A *timeline* é atualizada em tempo real.
Procedimento	1. Arrastar para a direita ou esquerda a *timeline*.

Identificador	UC12
Nome	Inspecionar carregamento de página.
Descrição Sumária	Permite ao utilizador visualizar uma descrição detalhada sobre o carregamento de uma página, nomeadamente os tempos do carregamento e também informação sobre todos os recursos descarregados.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de inspeção de uma sessão.
Pós-Condições	O utilizador visualiza todos os detalhes do carregamento.
Procedimento	1. Clicar em um circulo, correspondente a um carregamento de página.

Identificador	UC13
Nome	Ver informação de uma função.
Descrição Sumária	Permite ao utilizador obter informação sobre a função selecionada.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de inspeção de uma sessão.
Pós-Condições	O utilizador visualiza o tempo de execução da função, o seu corpo e cabeçalho.
Procedimento	1. Clicar em um quadrado, correspondente a uma função, na *timeline*.

Identificador	UC14
Nome	Inspecionar e certificar código da função.
Descrição Sumária	Permite ao utilizador, após selecionar uma função, analisar o seu código.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de inspeção de uma sessão. Ter selecionado uma função.
Pós-Condições	O utilizador obtém erros e possíveis ineficiências no código, bem como as soluções para os mesmos. Esta visualização, é feita num editor embutido no *browser*, onde é possível corrigir o código diretamente.
Procedimento	1. Clicar na opção Inspect Function.

Identificador	UC15
Nome	Inspecionar erro.
Descrição Sumária	Permite ao utilizador, verificar os detalhes do erro ocorrido, nomeadamente o seu código, mensagem e informações do seu ficheiro.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de inspeção de uma sessão.
Pós-Condições	O utilizador vê a informação sobre o erro selecionado.
Procedimento	1. Clicar num círculo correspondente a um erro, na *timeline*.

Identificador	UC16
Nome	Inspecionar evento HTML DOM.
Descrição Sumária	Permite ao utilizador, visualizar a informação do evento.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de inspeção de uma sessão.
Pós-Condições	O utilizador vê a informação sobre o evento.
Procedimento	1. Clicar num círculo correspondente a um evento, na *timeline*.

Identificador	UC17
Nome	Inspecionar pedido Ajax.
Descrição Sumária	Permite ao utilizador, verificar os detalhes do pedido ajax. Contém informação sobre duração, conteúdo da resposta, código, entre outros.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de inspeção de uma sessão.
Pós-Condições	O utilizador vê a informação detalhada sobre o pedido Ajax.
Procedimento	1. Clicar num círculo correspondente a um pedido Ajax, na *timeline*.

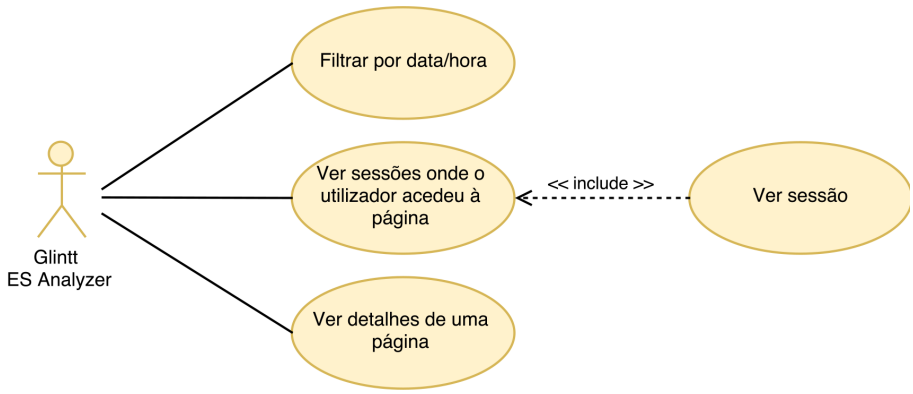

Figura 3.4: Diagrama de casos de utilização da área de visualização de páginas visitadas.

Identificador	UC18
Nome	Filtrar por data/hora.
Descrição Sumária	Permite ao utilizador verificar as páginas visitadas pelos clientes, num determinado espaço de tempo.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de páginas visitadas.
Pós-Condições	O utilizador vê um gráfico de barras onde cada uma corresponde a uma página e o seu valor ao número de visitas.
Procedimento	1. Clicar no calendário. 2. Selecionar uma opção predefinida ou se desejável, definir manualmente.

Identificador	UC19
Nome	Ver sessões onde o utilizador acedeu à página.
Descrição Sumária	Permite ao utilizador selecionar uma determinada página no gráfico para verificar quais as sessões onde esta foi visitada.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de páginas visitadas.
Pós-Condições	O utilizador tem acesso às sessões com acessos à página correspondente.
Procedimento	1. Clicar em uma barra do gráfico. 2. Clicar na opção View Sessions.

Identificador	UC20
Nome	Ver detalhes de uma página.
Descrição Sumária	Permite ao utilizador aceder à informação acerca dos tempos de carregamento e recursos de uma página.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de páginas visitadas.
Pós-Condições	O utilizador vê a informação de um carregamento de página.
Procedimento	1. Clicar em uma barra do gráfico.

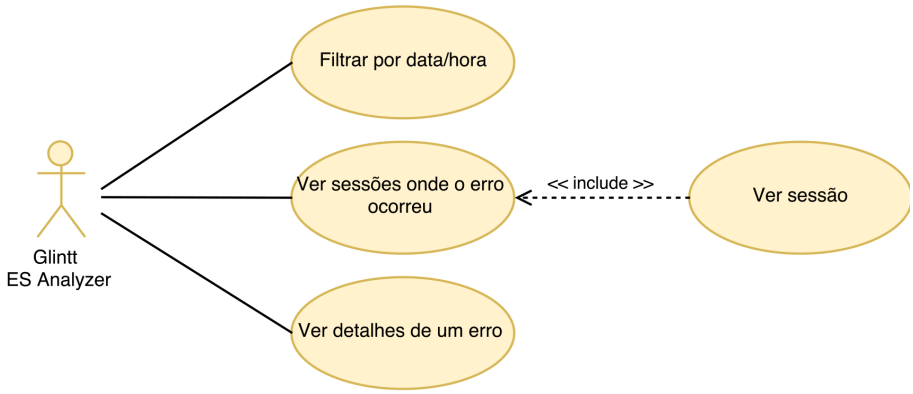

Figura 3.5: Diagrama de casos de utilização da área de erros ocorridos.

Identificador	UC21
Nome	Filtrar por data/hora.
Descrição Sumária	Permite ao utilizador verificar os erros que ocorreram num determinado espaço de tempo e quais os mais frequentes.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de inspeção de erros.
Pós-Condições	O utilizador vê um gráfico de dispersão que divide os vários tipos de erros.
Procedimento	1. Clicar no calendário. 2. Selecionar uma opção predefinida ou se desejável, definir manualmente.

Identificador	UC22
Nome	Ver sessões onde o erro ocorreu.
Descrição Sumária	Permite ao utilizador clicar num tipo de erro no gráfico, para verificar quais as sessões onde este ocorreu.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de inspeção de erros.
Pós-Condições	O utilizador tem acesso às sessões onde o erro aconteceu.
Procedimento	1. Clicar em um tipo de erro, no gráfico. 2.

Identificador	UC23
Nome	Ver detalhes de um tipo de erro.
Descrição Sumária	Permite ao utilizador ver em detalhe a informação acerca do erro.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de inspeção de erros.
Pós-Condições	O utilizador vê a informação sobre o erro.
Procedimento	1. Clicar em um tipo de erro, no gráfico.

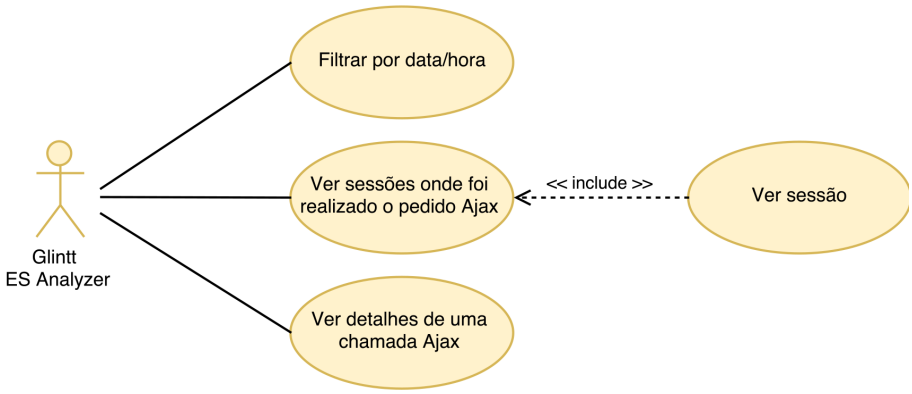

Figura 3.6: Diagrama de casos de utilização da área de analise de pedidos Ajax.

Identificador	UC24
Nome	Filtrar por data/hora.
Descrição Sumária	Permite ao utilizador ver um sumário sobre todos os pedidos Ajax que ocorreram num determinado espaço de tempo.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de análise Ajax.
Pós-Condições	O utilizador tem acesso às sessões desejadas.
Procedimento	1. Clicar no calendário. 2. Selecionar uma opção predefinida ou se desejável, definir manualmente.

Identificador	UC25
Nome	Ver sessões onde o foi realizado o pedido Ajax.
Descrição Sumária	Permite ao utilizador em que sessões foi realizado este tipo de pedido Ajax.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de análise Ajax.
Pós-Condições	O utilizador tem acesso às sessões.
Procedimento	1. Selecionar o pedido. 2. Clicar na opção Inspect Sessions.

Identificador	UC26
Nome	Ver detalhes de uma chamada Ajax.
Descrição Sumária	Permite ao utilizador ver informação sobre o pedido, como duração, conteúdo, resposta do servidor, entre outros.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de análise Ajax.
Pós-Condições	O utilizador tem acesso às sessões.
Procedimento	1. Selecionar o pedido Ajax.

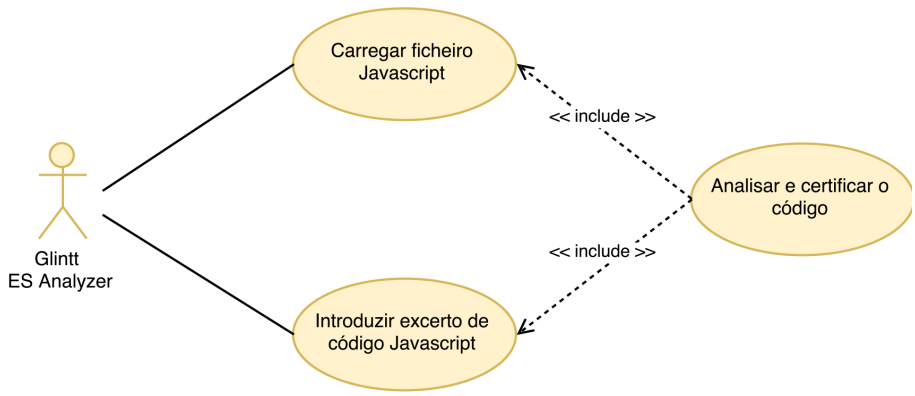

Figura 3.7: Diagrama de casos de utilização da área certificação de código Javascript.

Identificador	UC27
Nome	Carregar ficheiro Javascript.
Descrição Sumária	Permite ao utilizador carregar um ficheiro completo para o editor embutido na página.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de certificação de código Javascript.
Pós-Condições	O utilizador vê o código do ficheiro.
Procedimento	1. Clicar no botão Browse. 2. Selecionar o ficheiro desejado. 3. Clicar em abrir.

Identificador	UC28
Nome	Introduzir excerto de código Javascript.
Descrição Sumária	Permite ao utilizador escrever código no editor embutido na página, ou simplesmente copiar e colar.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de certificação de código Javascript.
Pós-Condições	O utilizador vê o código introduzido ou copiado.
Procedimento	Existem 2 Opções: 1. Escrever o código no editor. 1. Copiar o código e de seguida colar no editor.

Identificador	UC29
Nome	Analisar e certificar o código.
Descrição Sumária	Permite ao utilizador receber informação se o código contém algum problema ou ineficiência na sua construção. Se existir, é apresentado ao utilizador na linha correspondente a informação do problema e sugestões de resolução.
Ator	Engenheiro de Software - Glintt HS
Pré-Condições	Estar na área de certificação de código Javascript.
Pós-Condições	O utilizador certifica o código
Procedimento	1. Selecionar o pedido Ajax.

3.1.6 Project Charter

Figura 3.8: JSPerformance Certifier - Project Charter.

3.2 Arquitetura

3.2.1 Visão Geral

Esta solução é um sistema distribuído. Todos os componentes representados estão separados e são responsáveis por uma única função, comunicando entre si para conseguir obter e analisar a informação desejada.

No caso de não ser obrigatório obter métricas acerca da utilização de processador e memória, os módulos JSPCService e JSPCAnalyzer podem ser removidos, sem prejudicar o funcionamento da solução.

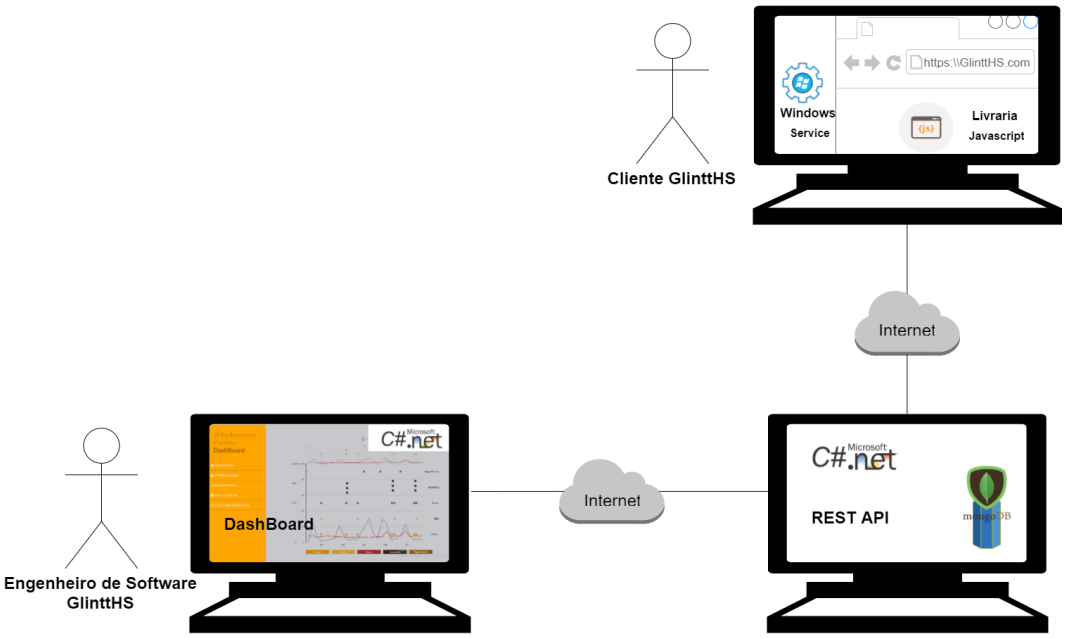

Figura 3.9: Visão geral do JSPerformance Certifier.

Na Figura 3.9 está representada uma visão bastante simplificada, mostrando apenas o essencial a alto nível, do sistema e do seu meio envolvente. O ponto de partida é a utilização das aplicações da empresa por parte dos clientes; Não só estará presente nesta aplicação a Biblioteca Javascript (JSPCRecorder), como também o serviço do Windows, instalado no computador onde esta é acedida. Toda a utilização é retratada e enviada para o serviço REST sendo armazenada por este na base de dados; Para terminar o processo, pode ser feita a análise dos dados na DashBoard.

Em seguida estes componentes estão explicados detalhadamente:

Bivraria Javascript (JSPCRecorder) conjunto de funções e código Javascript devidamente tratado e organizado numa biblioteca, que após a sua inclusão numa página web, regista toda a interação entre o utilizador e a aplicação.

Serviço Windows (JSPCService) componente essencial para o acompanhamento em *background* da aplicação web, nomeadamente com o objetivo de obter informação sobre o processador e memoria em utilização.

Aplicação Windows (JSPCAnalyzer) Aplicação Windows necessária para interagir com o ambiente de trabalho do utilizador e comunicar com o serviço. Esta aplicação é única e exclusivamente utilizada, pelo facto de os serviços do Windows funcionarem na Sessão 0 e não possuírem acesso ao ambiente dos utilizadores.

Servidor (JSPCRest) Este servidor REST recebe pedidos HTTP da biblioteca JSPCRecorder e da aplicação Windows, executando o tratamento necessário, para de seguida armazenar essa informação na base de dados.

Aplicação Web (JSPCDashboard) conjunto de interfaces com o intuito de oferecer uma representação gráfica e detalhada, de toda a informação que foi recolhida e armazenada pelos componentes anteriores.

3.2.2 Arquitetura Física

Na arquitetura física, está representada uma possível solução com 2 utilizadores ligados a uma máquina ao mesmo tempo. O número máximo de utilizadores é 254 por restrição do sistema operativo que apenas permite esse número de ligações remotas. A utilização do serviço do Windows e aplicação no ambiente de cada utilizador é necessária para recolher informação, relativa à utilização de memória e processador de um processo. A monitorização não é realizada pelo serviço do Windows por restrições do sistema operativo, que serão detalhadas no Capítulo 4. A comunicação entre as aplicações Windows e o serviço é realizada através de *Named Pipes*, não só devido à sua compatibilidade para com .NET Framework 3.5 e seguintes, mas também pelo facto de serem bastante eficientes.

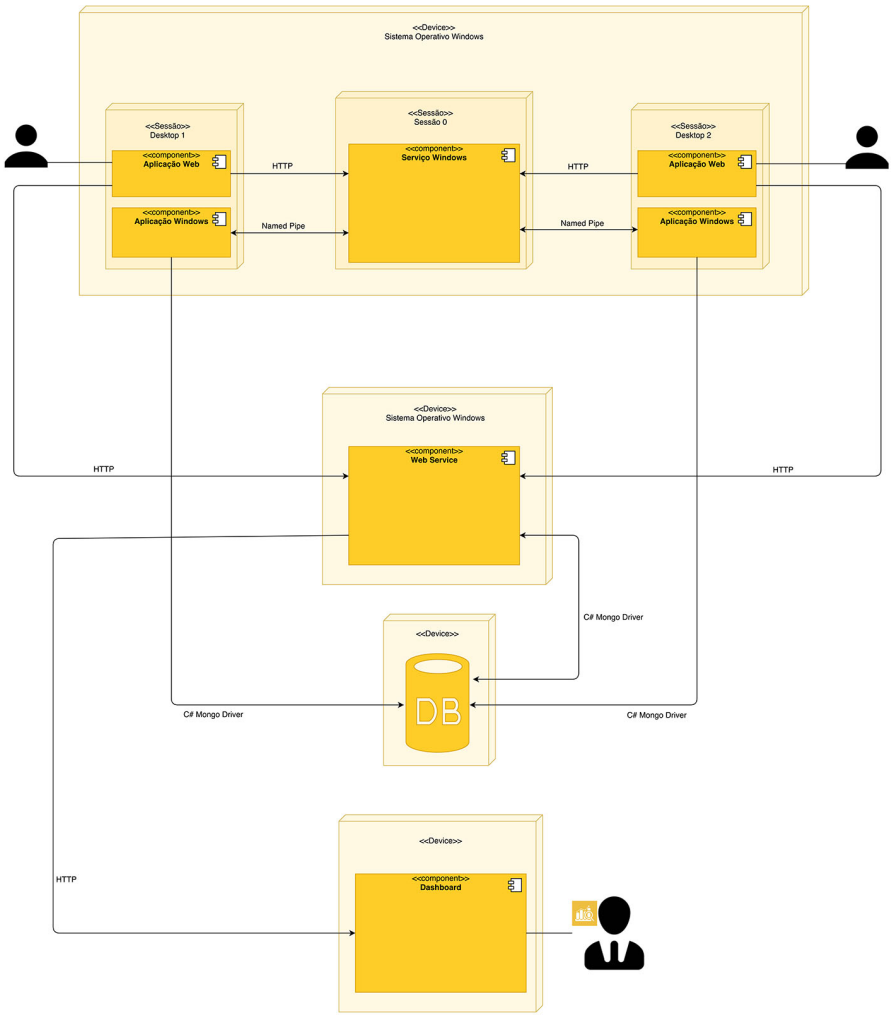

Figura 3.10: Diagrama de sequência S02.

3.2.3 Protocolo de comunicação

Existem dois protocolos de comunicação na solução proposta. O primeiro é necessário para a comunicação entre as aplicações web e o serviço do Windows, no mesmo computador. O segundo, é utilizado para consumir todas as comunicações com dados relativos às sessões que estão a ser monitorizadas. Ambos os serviços web utilizados nesta solução implementam a arquitetura REST e estão detalhados em seguida.

```
1     [Route("/GWS/{Id}/{Url*}", "GET")]
2     public class GlinttWebService
3     {
4         public string Id { get; set; }
5         public string Url { get; set; }
6     }
7     [Route("/GWS/Refresh/{Id}", "GET")]
8     public class GlinttWebServiceRefresh
```

```
9       {
10          public string Id { get; set; }
11      }
```

Listing 3.1: Código c# existente no serviço do Windows (JSPCService).

Na Listing 3.1 é possível verificar o código destinado a consumir comunicações por parte das aplicações web. Existem dois tipos de comunicação possíveis, uma delas relativa ao carregamento de uma página e outra para refrescamento ou abandono da mesma.

Id este campo representa o ID da sessão. Este ID é único e gerado para cada separador do *browser*.

Url endereço web da página que acabou de ser carregada. Esta variável é utilizada para consistência e minimização de erros.

O segundo protocolo de comunicação é utilizado, não só para consumir e guardar dados das sessões que estão a ser monitorizadas, mas também para recolher e fornecer informação da base de dados, para a DashBoard. Para todas as possíveis chamadas ao serviço, é sempre utilizado o mesmo formato que está representado na Listing 3.2 em seguinte.

```
1  {
2      "_id": null,
3      "start": 0,
4      "end": 0,
5      "summary": {
6          "eventCount": 0,
7          "errorCount": 0,
8          "ajaxCount": 0,
9          "funcCount": 0
10     },
11     "pageload": [
12         {
13             "time": 0,
14             "browserinfo": "",
15             "pageload": 0,
16             "dns": 0,
17             "tcp": 0,
18             "requestTime": 0,
19             "domTime": 0,
20             "pageUrl": "",
21             "resources": []
22         }
23     ],
24     "stream": {
25         "eventStack": [],
26         "errorStack": [],
27         "ajaxStack": [],
```

```
28          "funcStack": [],
29          "memoryStack": [],
30          "cpuStack": []
31      },
32      "IP": null
33  }
```

Listing 3.2: Exemplo de um objeto JSON vazio que representa uma sessão.

_id Id único de sessão.

start tempo de início da sessão.

end tempo final da sessão que corresponde à última comunicação efetuada para este ID.

summary sumário da sessão.

- **eventCount** número total de eventos que ocorreram na sessão.
- **errorCount** número total de erros que ocorreram na sessão.
- **ajaxCount** número total de chamadas Ajax que ocorreram na sessão.
- **funcCount** número total de funções executadas durante a sessão.

pageload lista com todos os objetos de "pageload", onde cada um representa o carregamento de uma página:

- **time** momento da ocorrência do carregamento da página web.
- **browserinfo** informação sobre tipo de *browser*, tipo de dispositivo e sistema operativo.
- **pageload** tempo total do carregamento.
- **dns** tempo usado apenas para executar a resolução do DNS.
- **tcp** tempo total da comunicação TCP com o servidor.
- **requestTime** tempo total da resposta do servidor.
- **domTime** tempo de carregamento do DOM.
- **pageUrl** endereço web da página em questão.
- **resources** lista de recursos carregados por esta página. Cada objeto contém os seguinte dados:
  - **type** tipo de recurso (script, CSS, image, etc).
  - **name** nome do recurso.
  - **totaltime** tempo total do carregamento do recurso.
  - **dnstime** tempo usado para executar a resolução do DNS.
  - **tcptime** tempo total da comunicação TCP com o servidor.

stream objeto que contém os *arrays* que retratam toda a interação do utilizador com a aplicação.

- **eventStack** lista de eventos ocorridos. Cada objeto tem as seguinte variáveis:
  - **time** momento da ocorrência do evento.
  - **name** nome do elemento do DOM onde é registado o evento.
  - **html** representa o HTML do elemento.
  - **handlers** *handlers* anexados ao elemento, caso existam.
    - **type** tipo de *handler* anexado ao elemento. Usualmente representa os *listeners* que lhe são atribuídos.
    - **function** função executada por este *handler*.
- **errorStack** lista com os erros ocorridos. Cada objeto tem as seguinte variáveis:
  - **time** momento da ocorrência do erro.
  - **page** endereço da página onde foi registado o erro.
  - **message** mensagem de erro.
  - **file** ficheiro onde o existe o problema de código.
  - **line** linha do ficheiro.
  - **col** coluna do ficheiro.
- **ajaxStack** lista com as chamadas Ajax efetuadas. Cada objeto tem as seguinte variáveis:
  - **time** tempo do início da chamada Ajax.
  - **url** endereço da chamada.
  - **type** tipo de chamada (Post, Get, Put, etc).
  - **contentType** formato do conteúdo (application/x-www-form-urlencoded, application/json, etc).
  - **responseText** resposta da chamada Ajax.
  - **status** código retornado.
  - **statusText** mensagem de texto sobre o estado da chamada.
  - **duration** tempo total de duração.
- **funcStack** lista com as funções executadas na sessão. Cada objeto tem as seguinte variáveis:
  - **time** momento do inicio da execução da função.
  - **duration** tempo de duração.
  - **funchead** cabeçalho da função que inclui variáveis desta execução específica.
  - **func** corpo da função.
- **memoryStack** lista com a informação acerca da utilização de memória por parte da aplicação (leitura efetuada a cada segundo). Cada objeto tem as seguinte variáveis:

  - **time** tempo da ocorrência da medição.
  - **value** valor da utilização de memória neste momento.
- **cpuStack** lista com a informação acerca da utilização de processador por parte da aplicação (leitura efetuada a cada segundo). Cada objeto tem as seguinte variáveis:
  - **time** tempo da ocorrência da medição.
  - **value** valor da utilização de processador neste momento.

Todas as possíveis rotas deste serviço estão pormenorizadas a seguir:

Serviço S01 - @POST RegisterSessionStart

Identificador	S01
Nome	RegistSessionStart
Descrição sumária	Envia um objeto de sessão com o id, o tempo inicial e final.
Requisitado por	Biblioteca Javascript
Entrada	_id, start, end
Saída	Código e mensagem de sucesso ou erro.

Tabela 3.8: Serviço S01.

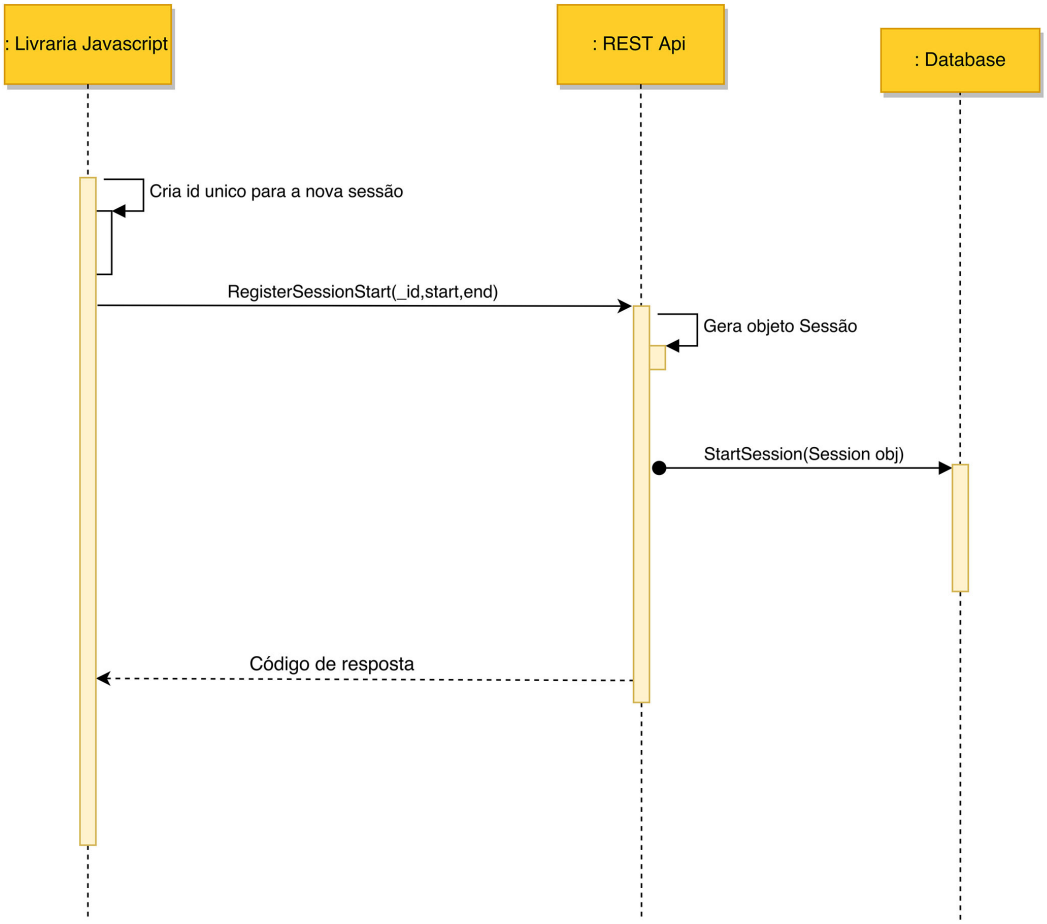

Figura 3.11: Diagrama de sequência S01.

Serviço S02 - @POST RegisterPageLoad

Identificador	S02
Nome	RegisterPageLoad
Descrição sumária	Envia um objeto de sessão com o id e um objeto de "pageload". Após introdução do objeto na lista correspondente, é atualizada a variável "end"para que o tempo final de sessão corresponda a este valor, até existir uma próxima comunicação.
Requisitado por	Biblioteca Javascript
Entrada	_id, objeto pageload, end
Saída	Código e mensagem de sucesso ou erro.

Tabela 3.9: Serviço S02.

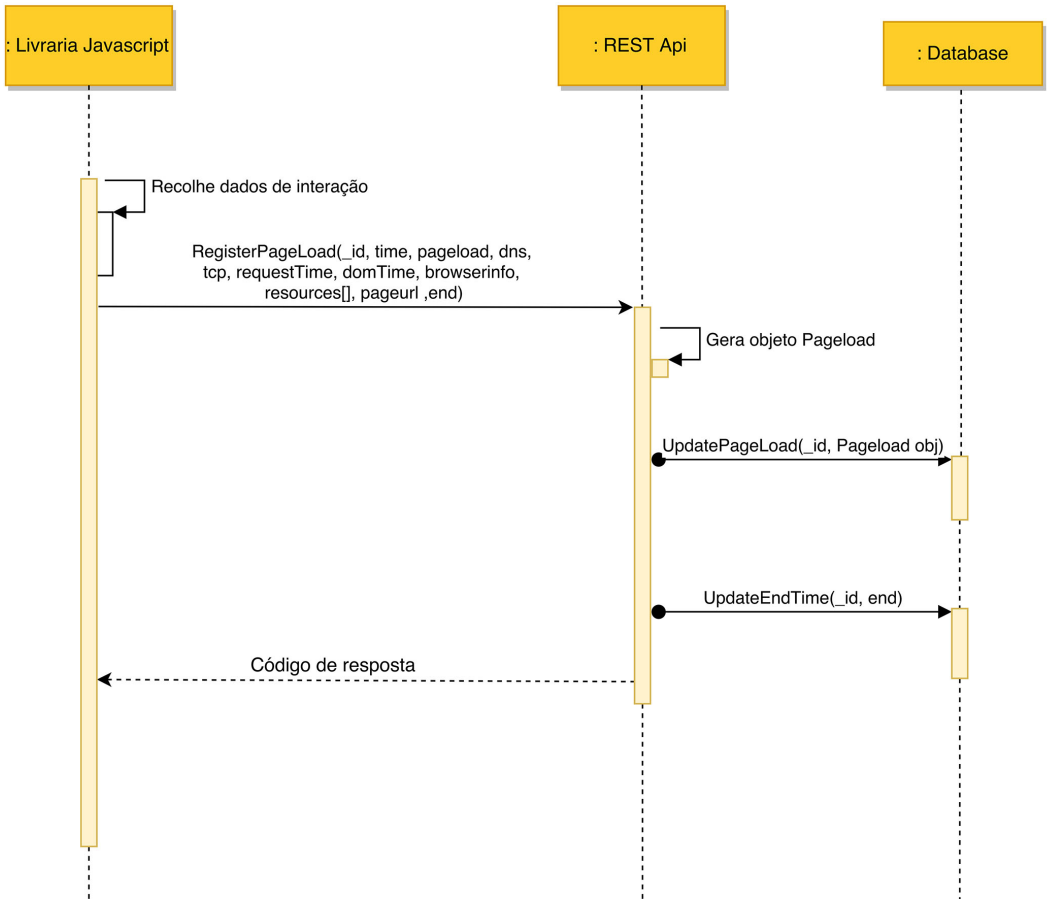

Figura 3.12: Diagrama de sequência S02.

Serviço S03 - @POST RegisterStream

Identificador	S03
Nome	RegisterStream
Descrição sumária	Envia um objeto de sessão com o id e o objeto de "stream", com todas as interações entre o utilizador e a aplicação que ainda não foram registadas. A variável "end"é também atualizada pela mesma razão do ponto anterior.
Requisitado por	Biblioteca Javascript ou Serviço Windows
Entrada	_id, objeto stream, end
Saída	Código e mensagem de sucesso ou erro.

Tabela 3.10: Serviço S03.

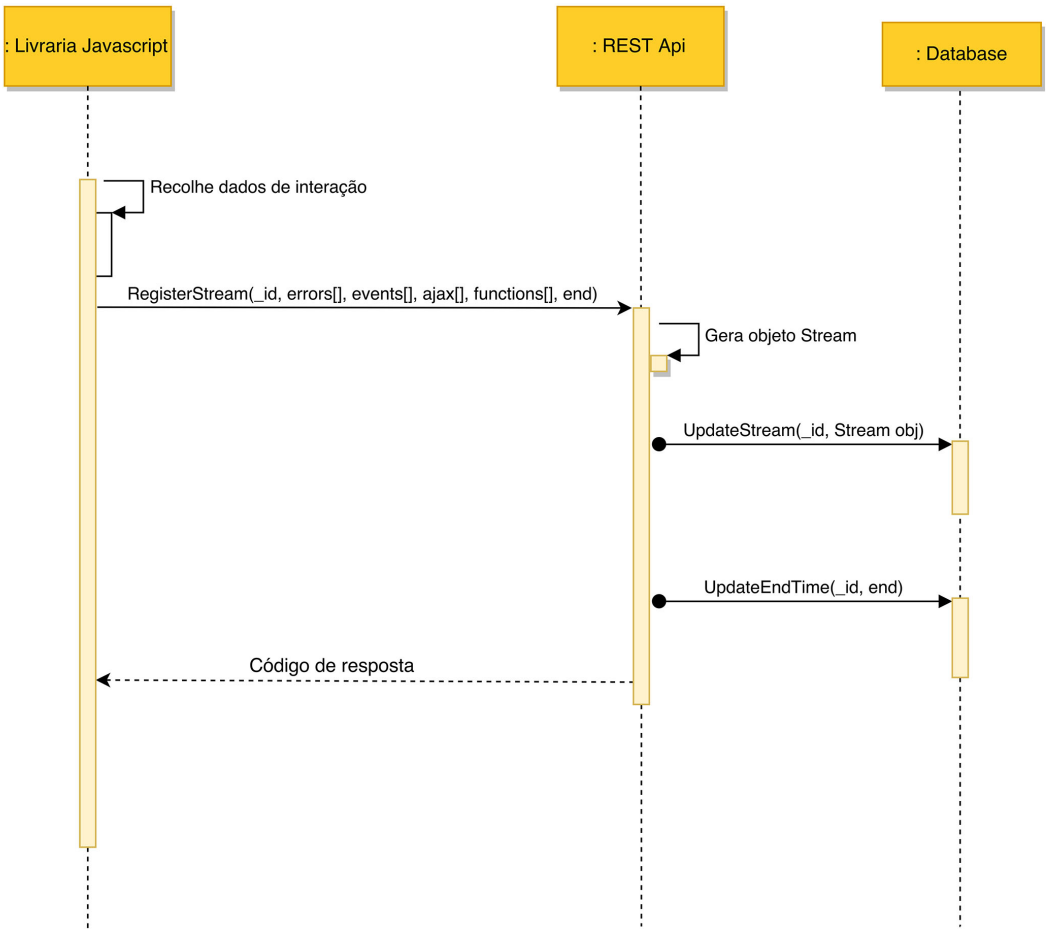

Figura 3.13: Diagrama de sequência S03.

Serviço S04 - @GET GetSessionsOnRange

Identificador	S04
Nome	GetSessionsOnRange
Descrição sumária	Retorna o conteúdo das sessões no espaço de tempo entre "start"e "end".
Requisitado por	Dashboard
Entrada	start, end
Saída	lista de sessões ("stream"e "pageload"não incluídos para não prejudicar performance).

Tabela 3.11: Serviço S04.

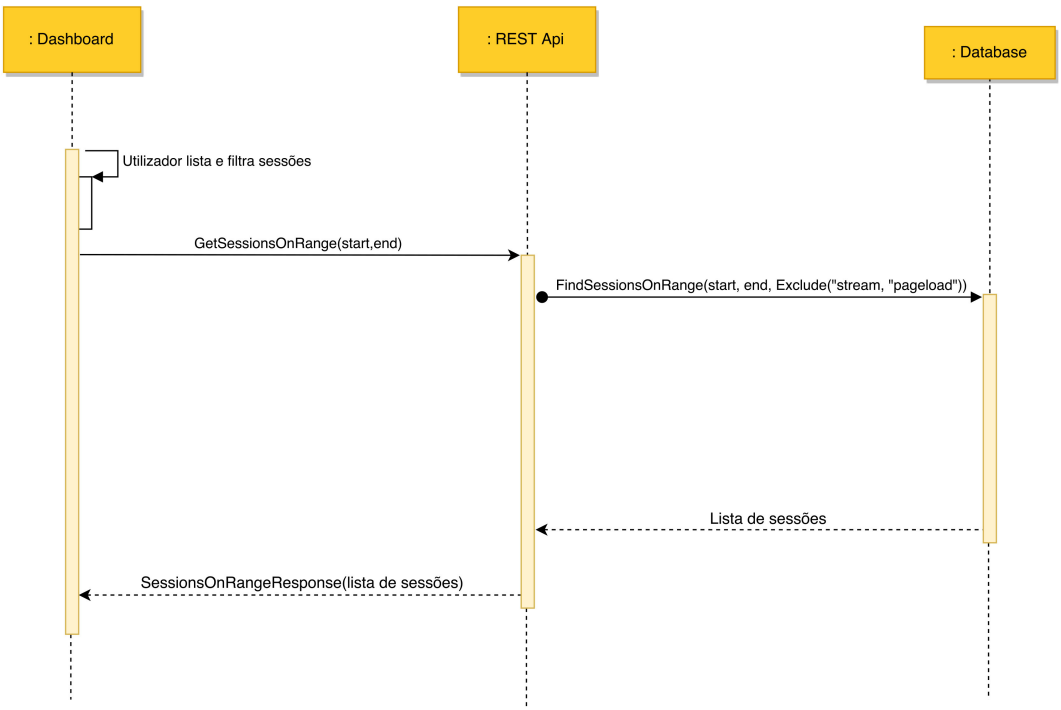

Figura 3.14: Diagrama de sequência S03 e S04.

Serviço S05 - @GET GetInfoFromSession

Identificador	S05
Nome	GetInfoFromSession
Descrição sumária	Retorna uma sessão com o id especificado.
Requisitado por	Dashboard
Entrada	_id
Saída	objeto de sessão.

Tabela 3.12: Serviço S05.

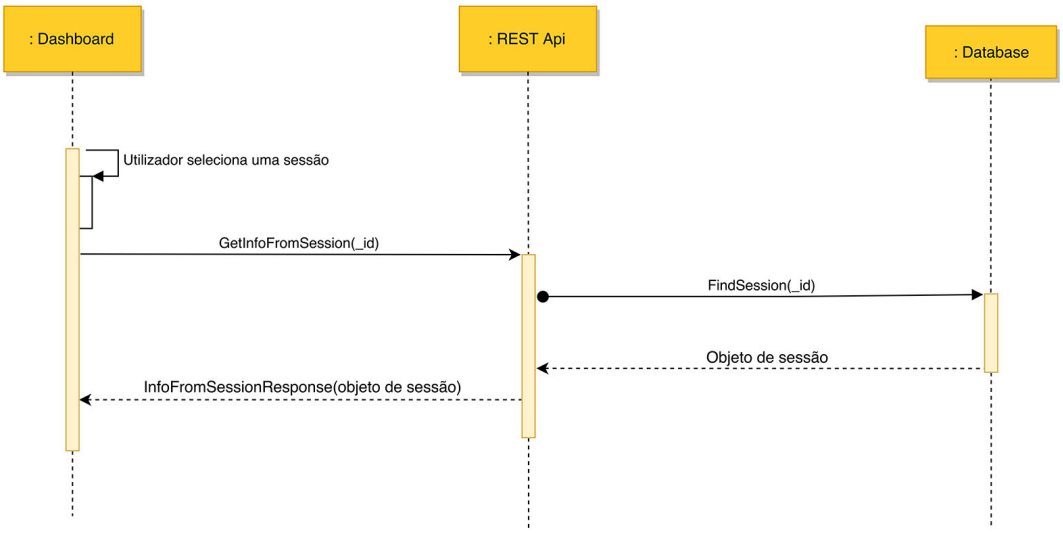

Figura 3.15: Diagrama de sequência S05.

Serviço S06 - @GET GetErrorsOnrange

Identificador	S06
Nome	GetErrorsOnrange
Descrição sumária	Obtém os erros das sessões entre "start"e "end".
Requisitado por	Dashboard
Entrada	start, end
Saída	lista com objetos de "erro".

Tabela 3.13: Serviço S06.

Serviço S07 - @GET GetPagesOnRange

Identificador	S07
Nome	GetAllPageLoads
Descrição sumária	Obtém os objetos de "pageload"das sessões entre "start"e "end".
Requisitado por	Dashboard
Entrada	start, end
Saída	lista com objetos de "pageload".

Tabela 3.14: Serviço S07.

Serviço S08 - @GET GetAjaxOnRange

Identificador	S08
Nome	GetAjaxOnRange
Descrição sumária	Obtém as chamadas Ajax de todas as sessões entre "start"e "end".
Requisitado por	Dashboard
Entrada	start, end
Saída	lista com objetos "ajax".

Tabela 3.15: Serviço S08.

Serviço S09 - @GET GetFunctionsOnRange

Identificador	S09
Nome	GetFunctionsOnRange
Descrição sumária	Obtém as funções executadas nas sessões entre "start"e "end".
Requisitado por	Dashboard
Entrada	start, end
Saída	lista com objetos de "função".

Tabela 3.16: Serviço S09.

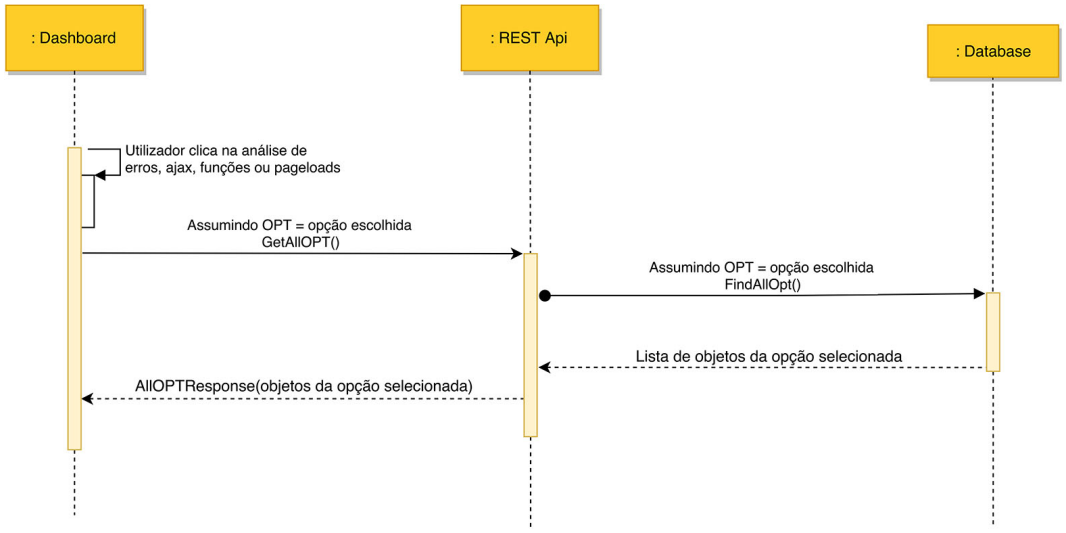

Figura 3.16: Diagrama de sequência S06, S07, S08 e S9.

Capítulo 4

Implementação

Este capítulo é dedicado aos detalhes da solução proposta. São apresentados também os resultados obtidos após a incorporação do sistema com uma aplicação da Glintt HS.

4.1 Detalhes de Implementação

Nesta secção são expostos os detalhes de desenvolvimento dos 5 artefactos, bem como todas as restrições que foram encontradas durante o seu desenvolvimento e as respetivas soluções implementadas.

4.1.1 Tecnologias

Figura 4.1: Tecnologias utilizadas pelos artefactos.

Foi escolhido a utilização de Microsoft .NET e Visual C# para interagir com o SO no serviço e aplicação Windows, uma vez que, para acompanhar as aplicações do cliente apenas existe a garantia que estes têm instaladas estas tecnologias. Também a sua utilização foi requerida para o servidor REST visto serem as tecnologias utilizadas pela empresa.

A biblioteca foi desenvolvida única e exclusivamente através de código Javascript.

Em relação à Dashboard, foi utilizada a arquitetura MVC presente na .NET Framework. Esta arquitetura permite estruturar a aplicação nas seguintes camadas:

Models Os modelos são responsáveis pelo manipulação de dados. São normalmente utilizados para definir objetos.

Views É a camada de interação com o utilizador. Esta apenas faz a visualização de dados através de HTML ou XML.

Controllers O controlador é responsável por atender a todas as requisições do utilizador, trabalha com os modelos e define o que é apresentado.

A nível de *Back-End* (Modelos e Controladores), esta arquitetura utiliza a linguagem C#.

Para o *Front-End* nomeadamente para as *Views*, as tecnologias utilizadas são as usuais: Javascript, HTML e CSS e Razor.

A criação de gráficos para a amostragem da informação foi realizada em D3.js, uma biblioteca para manipular e apresentar dados através de Javascript CSS e HTML.

No ecrã de análise de código, para gerar a árvore de ficheiros foi utilizada a linguagem VB e para analisar a sua complexidade utilizado NodeJS.

Por fim, para armazenar e manipular toda a informação, foi escolhido o tipo de base de dados MongoDB.

4.1.2 Biblioteca Javascript - JSPCRecorder

Esta biblioteca é composta por funções e variáveis Javascript que em conjunto monitorizam toda a interação entre o utilizador e uma página HTML. Nesta secção vão ser explicadas em detalhe todas as funcionalidades e pormenores relevantes para o seu funcionamento. Convém salientar que uma sessão corresponde a um separador do *browser*, ou seja uma sessão pode englobar diversas visitas a páginas distintas desde que estas contenham a biblioteca incluída, caso contrário será inicializada uma sessão diferente após um período de tempo.

4.1.2.1 Requisitos

Para esta biblioteca foi requisitada a compatibilidade para os seguintes *browsers*:

- Internet Explorer 8
- Internet Explorer 9
- Internet Explorer 10
- Internet Explorer 11
- Microsoft Edge
- Google Chrome 45+

4.1.2.2 Início de sessão

Todo o processo de monitorização é iniciado no momento em que o utilizador entra na página. É então criado um ID único que é associado à *tag* "body"do documento, e armazenado no *browser* através da API *sessionStorage* que guarda a informação durante o tempo de vida de uma sessão, ou seja, até ao momento em que o separador é removido. De seguida, é realizada a comunicação com o serviço do Windows, informando que uma nova página HTML com esta biblioteca foi acedida para que este despolete à sua monitorização; Por fim, são enviados os dados para o servidor com a finalidade de iniciar uma nova sessão e registar a mesma na base de dados.

Para gerar um ID único, foi utilizada a função descrita na Listing 4.1 que utiliza números aleatórios organizados num formato que segue a norma RFC4122 version 4.

```
1  function generateUUID(){
2      var d = new Date().getTime();
3
4      var uuid = 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function(c)
           {
5          var r = (d + Math.random()*16)%16 | 0;
6          d = Math.floor(d/16);
7          return (c=='x' ? r : (r&0x3|0x8)).toString(16);
8      });
9      return uuid;
10 }
```

Listing 4.1: Função Javascript utilizada para gerar o ID único de sessão.

Devido à possível colisão de ID's. a função substitui os 13 primeiros dígitos por uma parte do *timestamp* obtido pela função *getTime* do objeto *Date*, que corresponde ao tempo em milisegundos desde 1 Janeiro 1970 00:00:00 UTC.

Após o início de sessão a biblioteca fica à escuta de eventos, erros, pedidos Ajax e por último funções executadas, até o utilizador decidir abandonar o separador. Todos estes dados são enviados por pedidos HTTP para o servidor REST que faz o seu tratamento e registo na sessão iniciada previamente.

4.1.2.3 Comunicação

Para enviar pedidos HTTP para o servidor e para o serviço do Windows, foi utilizada a API XHR. Devido a problemas de compatibilidade, o Internet Explorer nas versões 8 e 9 não permite a sua utilização e obriga ao uso do objeto *XDomainRequest*, pelo que foi necessário adaptar as funções para utilizar o objeto correto consoante o tipo e versão do *browser*.

```
1  function doGetRequest(url) {
2      // Create the XHR object.
3      var xhr = new XMLHttpRequest();
```

```
    if ("withCredentials" in xhr) {
        // XHR for Chrome/Firefox/Opera/Safari.
        xhr.open('get', url, true);
    } else if (typeof XDomainRequest != "undefined") {
        // XDomainRequest for IE.
        xhr = new XDomainRequest();
        xhr.open('get', url);
    } else {
        // CORS not supported.
        xhr = null;
    };
    if (!xhr) {
        return;
    };
    // Response handlers.
    xhr.onload = function () {
    };
    xhr.onerror = function () {
    };
    xhr.onprogress = function () { };
    xhr.ontimeout = function () { };
    setTimeout(function () {
        xhr.send();
    }, 0);
};
```

Listing 4.2: Função Javascript utilizada para enviar pedidos HTTP GET para o servidor.

```
function doPostRequest(url, data) {
    // Create the XHR object.
    var xhr = new XMLHttpRequest();
    if ("withCredentials" in xhr) {
        // XHR for Chrome/Firefox/Opera/Safari.
        xhr.open('post', url, true);
    } else if (typeof XDomainRequest != "undefined") {
        // XDomainRequest for IE.
        xhr = new XDomainRequest();
        xhr.open('post', url);
    } else {
        // CORS not supported.
        xhr = null;
    };
    if (!xhr) {
        return;
    };
    // Response handlers.
    xhr.onload = function () {
        eventStack.length = 0;
```

```
21          errorStack.length = 0;
22          ajaxStack.length = 0;
23          memoryStack.length = 0;
24          cpuStack.length = 0;
25          funcStack.length = 0;
26      };
27      xhr.onerror = function () {
28      };
29      xhr.onprogress = function () { };
30      xhr.ontimeout = function () { };
31      setTimeout(function () {
32          xhr.send(JSON.stringify(data));
33      }, 0);
34  };
```

Listing 4.3: Função Javascript utilizada para enviar dados por HTTP POST para o servidor.

É necessário dar especial atenção à utilização das funções *onload* e *onerror* com o corpo vazio e à função *send* dentro de um *setTimeout* de 0 segundos. A sua utilização, apesar de invulgar, é obrigatória para o Internet Explorer 9, caso contrário o envio resulta em erro [1].

4.1.2.4 Eventos

Um evento representa uma interação do utilizador com um elemento do DOM. Para sua captura é necessário adicionar um *listener* ao documento para que este fique à escuta. Estes eventos podem ser do seguintes tipos:

- eventos de rato
- eventos de teclado
- eventos de drag&drop
- eventos de formulário
- eventos de impressão
- eventos de clipboard
- eventos de touch
- eventos relacionados com média
- eventos relacionados com animações

Cada tipo referido acima engloba um conjunto de eventos específicos que podem ser consultados na página do W3Schools [2].

[1] http://cypressnorth.com/programming/internet-explorer-aborting-ajax-requests-fixed/
[2] http://www.w3schools.com/jsref/dom_obj_event.asp

```
var eventTracking = function (ev) {
    var elem = new Object();
    if (ev.srcElement) elem = ev.srcElement;
    else if (ev.target) elem = ev.target;
    var events = Array();
    if ($(elem).data("events")) {
        $.each($(elem).data("events"), function (i, e) {
            $.each(e, function (j, h) {
                var event = {
                    "type": h.type,
                    "function": h.handler.toString()
                };
                events.push(event);
            });
        });
        elem.time = Date.now();
        elem.events = events;
        _this.capture(elem, "EVENT");
    }
};
document.addEventListener("click", eventTracking, false);
```

Listing 4.4: Função Javascript utilizada para capturar a ocorrência de um evento.

Na Listing 4.4 está um exemplo da captura de eventos de "click"do rato.

4.1.2.5 Erros

Para ter informação sobre os erros que ocorrem, é utilizado o método *onerror* que nos oferece todos os dados relevantes.

```
window.onerror = function (errorMsg, url, lineNumber, column, errorObj) {
    var err = new Object();
    err.message = errorMsg;
    err.filename = url;
    err.lineno = lineNumber;
    err.colno = column;
    err.time = Date.now();
    _this.capture(err, "ERROR");
};
```

Listing 4.5: Função que captura os erros que ocorrem na página HTML.

4.1.2.6 Ajax

Relativamente às chamadas Ajax, o início do pedido é capturado pelo método *ajaxStart* do documento onde é registado o *timestamp* e por fim é utilizado o método *ajaxComplete* onde esse *timestamp* é subtraído para saber o tempo total da execução. O método *ajaxComplete* oferece todas as restantes informações que são armazenadas no objeto de sessão detalhado na Secção 3.2.3.

```
1  $(document).ajaxStart(function (event, request, settings) {
2      start = event.timeStamp;
3  });
4  $(document).ajaxComplete(function (event, request, settings) {
5      end = event.timeStamp;
6      var ajax = new Object();
7      ajax.url = settings.url;
8      ajax.type = settings.type;
9      ajax.contentType = settings.contentType;
10     ajax.responseText = request.responseText;
11     ajax.status = request.status;
12     ajax.statusText = request.statusText;
13     ajax.duration = end - start;
14     ajax.time = Date.now();
15     _this.capture(ajax, "AJAX");
16 });
```

Listing 4.6: Funções jQuery que permitem obter o início e fim de um pedido Ajax.

4.1.2.7 Funções

Para conseguir obter os dados de uma função, a biblioteca percorre o objeto *window* e seus componentes, adicionando um pedaço de código extra a todas as funções, o que permite obter o tempo de duração da sua execução, o cabeçalho e o corpo da função.

```
1  function addLogToFunc(func, name, namespace) {
2      return function () {
3          var start;
4          if (window.performance.now)
5              start = window.performance.now();
6          else
7              start = Date.now();
8          var start_date = Date.now();
9          // Executes the function normally
10         var result = func.apply(this, arguments);
11         var end;
12         if (window.performance.now)
13             end = window.performance.now();
14         else
```

```
15              end = Date.now();
16          var funchead = namespace + '.' + name + '(';
17          for (var i = 0; i < arguments.length; i++) {
18              if (i > 0)
19                  funchead += ', ';
20              funchead += JSON.stringify(arguments[i]);
21          }
22          funchead += ');';
23          var funcobj = new Object();
24          var temp = func;
25          funcobj.time = start_date;
26          funcobj.duration = (end - start);
27          funcobj.funchead = funchead;
28          funcobj.func = temp.toString();
29          funcStack.push(funcobj);
30          return result;
31      }
32  };
33  function findFuncInNamespace(namespaceObject, namespace) {
34      for (var name in namespaceObject) {
35          var func = namespaceObject[name];
36          if (Object.prototype.toString.call(potentialFunction) === '[object Function
                ]') {
37              if (namespace !== 'window' || (pageNameSpaces.indexOf(name) > -1 &&
                    name.indexOf('$') === -1 && name.indexOf('jQuery') === -1 && name.
                    indexOf('detect') === -1 && name.indexOf('recorder') === -1)) {
38                  namespaceObject[name] = addLogToFunc(func, name, namespace);
39              }
40          }
41      }
42  };
```

Listing 4.7: Método Javascript utilizado para adicionar código a uma função.

As duas funções visíveis na Listing 4.7 são utilizadas em conjunto, para recolher informação sobre todas as execuções que ocorrem durante a utilização da página HTML. Para não existir informação redundante, é ignorado no início do carregamento da página, todos os objetos e funções utilizadas por uma página vazia. Além disso são também ignoradas as funções relacionadas com *jQuery* e esta biblioteca.

4.1.2.8 Refrescamento de uma página

Durante o desenvolvimento desta biblioteca foi imediatamente identificado o problema dos dados que ainda não foram armazenados, desaparecerem no momento em que o utilizador abandona a página ou simplesmente navega para outra. Como todas as interações são armazenadas em *arrays* Javascript, foi inicialmente ponderado definir um número estático de elementos máximos que um *array* poderia conter, sendo que após esse valor ser atingido os dados seriam enviados

para o servidor. Como esta solução não era a melhor, o envio da informação para o servidor foi alterado de modo a que, a informação fosse enviada periodicamente, de 10 em 10 segundos. Para garantir ainda menos perdas de informação, é utilizado o método *onbeforeunload* que captura o momento de descarregamento da aplicação. Este método dispara no caso de uma nova navegação, um refrescamento da página ou o seu abandono. Assim, é possível garantir que não existe qualquer perda de informação.

4.1.2.9 Problemas encontrados

Como este componente foi desenvolvido com o propósito de ser utilizado apenas por quem desenvolve as aplicações web da Glintt, a compatibilidade não era uma questão pertinente. Esta situação foi alterada após surgir o objetivo de incluir a biblioteca para monitorizar as aplicações não só, durante o desenvolvimento, mas também durante a sua utilização.

Em seguida é apresentada uma lista dos problemas de compatibilidade que foram encontrados para alguns *browsers* em específico.

Internet Explorer 8

- Date.now() - Método que retorna o tempo em milissegundos desde 1 Janeiro 1970 00:00:00 UTC.
- Array.indexOf(searchElement, index) - Retorna o índice onde o elemento é encontrado no *array*.
- Array.filter(callback) - Este método cria um novo *array* com todos os elementos que passam a condição.
- EventTarget.addEventListener(type, func) - Regista um *listener* de um evento especifico no elemento.
- removeEventListener(type, func) - Remove o *listener* de um elemento.
- Navigation Timing API - API detalhada na Secção 2.1.1.
- Resource Timing API - API detalhada na Secção 2.1.2.

Internet Explorer 9

- Resource Timing API - API detalhada na Secção 2.1.2.

Todos estes problemas de compatibilidade foram contornados com a utilização de *polyfills*, com a exceção das API's *Resource Timing* e *Navigation Timing*, porque estas tem de ser implementadas pelo próprio *browser*. *Polyfill* é um termo atribuído a um de excerto de código que oferece as funcionalidades desejadas em caso do *browser* não as suportar.

4.1.3 Serviço Windows - JSPCService

Como foi explicado no Capitulo 2, obter informação sobre a utilização de memória e processador no lado do cliente, não é exequível. No entanto, surgiu a possibilidade de instalar serviços do Windows não só nos dispositivos da empresa, mas também nos dispositivos dos clientes. Através da sua utilização estas métricas deixam de ser impossíveis de recolher.

4.1.3.1 Requisitos

Devido à versatilidade de dispositivos dos clientes da Glintt HS, a compatibilidade foi uma questão logo de início abordada. Em seguida é apresentada uma lista de versões Windows para as quais foi necessário desenvolver o serviço.

- Windows XP
- Windows Server 2003
- Windows Server 2008
- Windows Server 2012
- Windows 7
- Windows 8
- Windows 8.1
- Windows 10

Para conseguir corresponder às necessidades, o serviço foi desenvolvido para .Net Framework 3.5, última versão suportada por Windows XP e Server 2003.

Além da questão da compatibilidade, surgiu durante o desenvolvimento a necessidade de suportar também ligações remotas, em que numa máquina podem estar conectados múltiplos clientes.

4.1.3.2 Restrições

A principal restrição de um serviço é o isolamento dos mesmos na Sessão 0, apesar de ser extremamente importante a nível de segurança. Após o Windows XP e Windows Server 2003 todos os serviços foram isolados na sessão 0 e as aplicações do ambiente de utilizador deixam de ter acesso direto a estes e vice versa por questões de segurança, nomeadamente para prevenir *shattered attacks*, *DDOS attacks* e execução de código malicioso das aplicações com privilégios elevados.

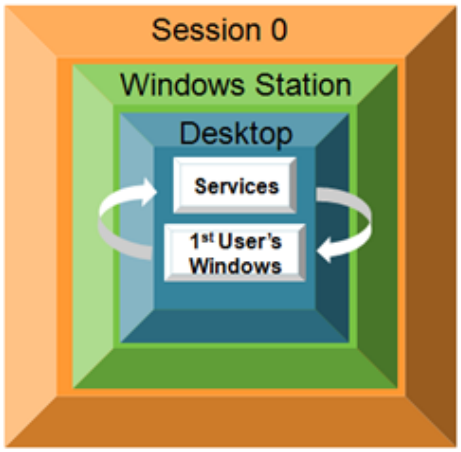

Figura 4.2: Serviços no Windows XP e Server 2003.

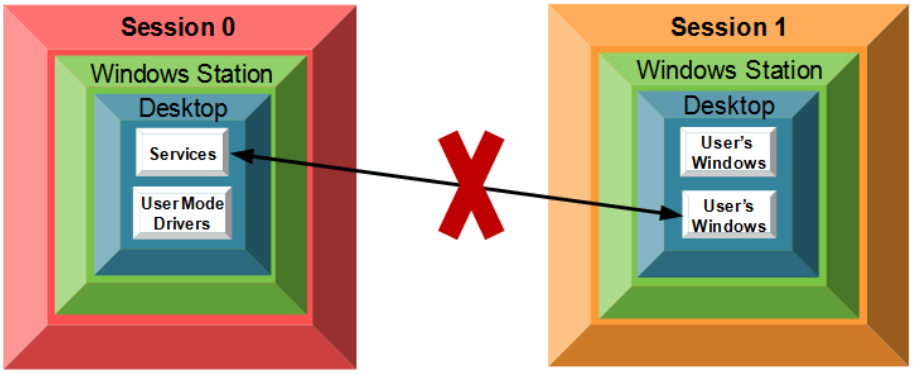

Figura 4.3: Serviços isolados na Sessão 0.

As Figuras 4.2 e 4.3 retratam a alteração efetuada [3]. Com este isolamento, deixa de ser possível ao serviço identificar algo relacionado com o ambiente interativo do utilizador, o que para este caso específico foi prejudicial pois era necessário adquirir informação sobre os separadores, conteúdo HTML, entre outros.

4.1.3.3 Funcionamento

O objetivo do serviço do Windows é acompanhar uma aplicação web e reportar a utilização de memória e processador. Para isso é fulcral saber qual o processo responsável pelo separador do *browser*.

O procedimento não pode ser executado unicamente pelo serviço, porque este não tem acesso ao ambiente dos utilizadores como foi explicado na secção anterior. Para tal, foi necessário criar uma aplicação Windows que interagisse com o serviço. Como é imposto suportar ligações remotas, esta aplicação é executada pelo serviço no ambiente de cada utilizador, no momento do seu *login*. Os detalhes de implementação desta aplicação serão explicados na Secção 4.1.4.

[3] http://blogs.technet.com/b/askperf/archive/2007/07/24/sessions-desktops-and-windows-stations.aspx

Para melhor compreender a sequência de eventos, é apresentado em seguida uma lista com os acontecimentos ordenados, desde momento em que a página HTML é acedida, até ao momento em que a sua utilização memoria e processador são acompanhados.

1. A página HTML comunica com o serviço por um pedido HTTP GET com o ID da sessão e o URL da aplicação.
2. O serviço descobre qual o *browser* através do *parsing* da string *UserAgent* do pedido HTTP.
3. O serviço avisa a aplicação no ambiente do utilizador, ou as várias aplicações em caso de sessões remotas, que uma nova página foi acedida.
4. O processo responsável pelo separador começa a ser monitorizado e a utilização de memória e processador é registada na sessão com o ID fornecido.

4.1.3.4 Comunicação

Neste serviço existem dois protocolos de comunicação, a comunicação proveniente das páginas HTML e a comunicação realizada com as aplicações dos ambientes de utilizador.

No primeiro caso, foi integrado um serviço web explicado na Secção 3.2.3 no serviço Windows para que este possa receber comunicações através de Javascript.

Para garantir a estabilidade deste serviço Web, foram realizados testes com 1000 *Threads* a executar 2 chamadas por cada 100 milissegundos. Os resultados estão visíveis na Figura 4.4.

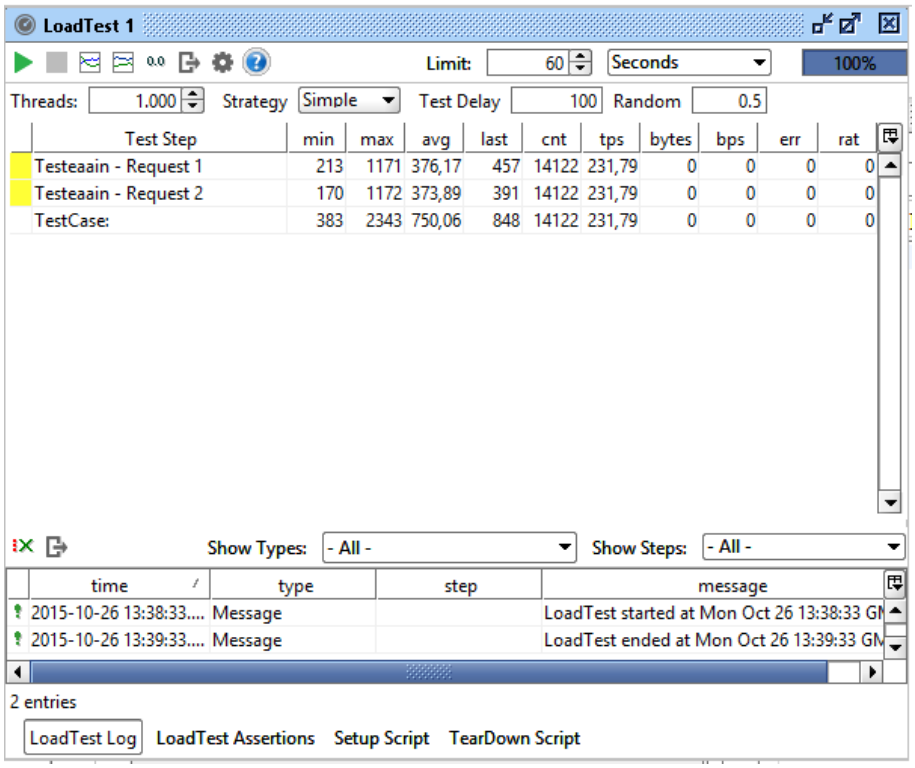

Figura 4.4: Test ao serviço web.

No segundo caso, para realizar a comunicação entre o serviço e a aplicação do ambiente de utilizador é usado um método de *Inter Process Communication* [4]. Apesar de *Memory Mapped Files* serem mais eficientes devido à sua leitura e escrita serem feitas diretamente na memória, estes apenas estão disponíveis a partir da versão 4.5 da .Net Framework. Por esta razão foi escolhido a utilização de *Named Pipes*, não só pela sua compatibilidade, mas também pela sua fácil compreensão, adaptação e eficiência.

4.1.4 Aplicação Windows - JSPCAnalyzer

Este componente tem como objetivo acompanhar a informação em tempo real, de todos os separadores que estão ativos no SO e proceder à sua monitorização quando preciso.

4.1.4.1 Requisitos

Os requisitos do serviço Windows aplicam-se também nesta aplicação.
Foi requerido o funcionamento desta aplicação para os seguintes browsers:

- Internet Explorer 8
- Internet Explorer 9
- Internet Explorer 10
- Internet Explorer 11
- Microsoft Edge
- Google Chrome 40+

4.1.4.2 Funcionamento

A aplicação é executada no momento em que o utilizador faz *login* no sistema; De seguida é realizada a conexão do *Named Pipe* ao serviço para ficar à escuta de comunicações; Após informação de uma nova página, esta aplicação irá verificar os separadores que está a acompanhar; A monitorização é feita, caso exista um separador com a página HTML indicada.

Um *browser* pode ser acompanhado através da sua automação. Este procedimento de acompanhamento é complexo e varia consoante a versão, portanto cada um será explicado em detalhe separadamente.

4.1.4.3 Internet Explorer

Relativamente ao Internet Explorer, existem 2 bibliotecas presentes no Windows que facilitam a sua automação:

[4] https://msdn.microsoft.com/en-us/library/windows/desktop/aa365574(v=vs.85).aspx

shdocvw.dll (Microsoft Internet Controls)

mshtml.dll Microsoft HTML Object Library

Em conjunto, a sua utilização permite acompanhar todas as instâncias do Internet Explorer e aceder ao seu HTML, DOM e também a toda a informação relativamente ao processo e janela responsáveis por cada uma delas.

Para este *browser* não existe a necessidade de procurar informação sobre quais os separadores ativos, porque a primeira biblioteca já o faz. Portanto para proceder à monitorização de um separador é executado o seguinte processo:

1. Obter o objeto *ShellWindows* responsável por armazenar toda a informação das instâncias.

2. Filtrar a lista de instâncias por URL, para aumentar a eficiência e diminuir tempo de execução.

3. Para cada uma, aceder ao documento e verificar o atributo "id"da *tag* "body".

4. Quando encontrado, o processo responsável pelo separador começa a ser monitorizado.

No caso do Internet Explorer, como o acompanhamento é feito pelas próprias bibliotecas do Windows, não existe qualquer possibilidade de falha independentemente do número de utilizadores ligados ou separadores abertos. Mesmo assim, foram realizados testes exaustivos através do lançamento de 40 separadores, no mesmo milissegundo, com a mesma aplicação. Em todos os testes foi alcançada uma taxa de sucesso de 100%. É de realçar que não existe qualquer problema no caso de sessões remotas, porque esta *framework* apenas tem acesso aos elementos do seu ambiente de utilizador.

4.1.4.4 Microsoft Edge

No caso do Microsoft Edge, apesar de pertencer ao Windows da mesma forma que o Internet Explorer, não existe a opção de utilizar as mesmas bibliotecas explicadas anteriormente. Para realizar a sua automação recorreu-se à *framework* de acessibilidade do Windows *uiAutomation* que disponibiliza o acesso aos elementos interativos do ambiente de trabalho.

Antes de mais, convém saber como é que os elementos são organizados e desenhados pelo Microsoft Edge.

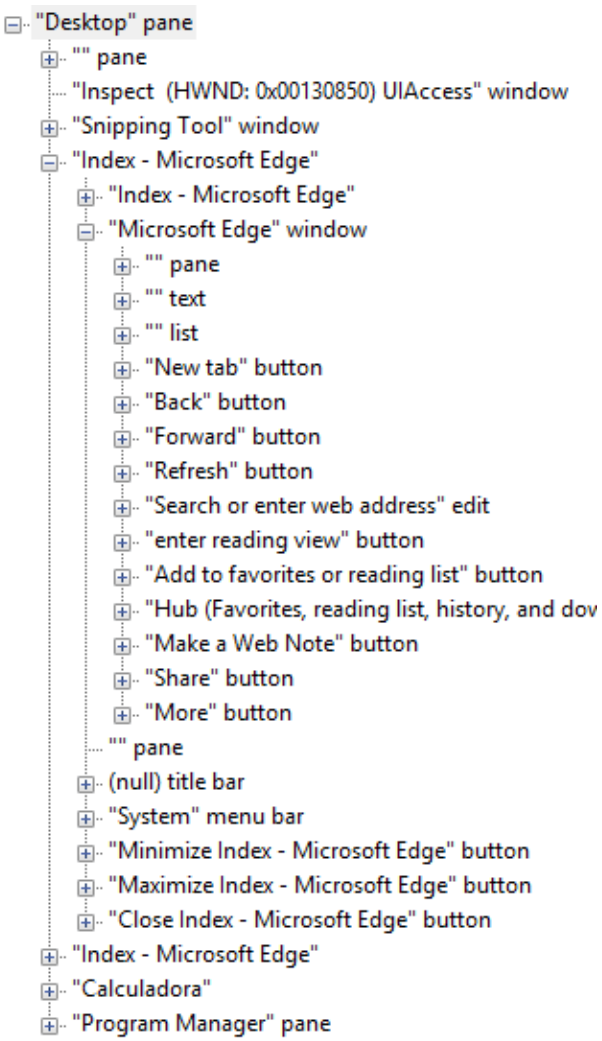

Figura 4.5: Árvore de Automação do Windows.

Na Figura 4.5 está representada uma árvore de automação do Windows, obtida com a ferramenta "Inspect.exe"do Windows Kits, onde estão presentes duas janelas do Microsoft Edge com dois separadores cada e uma calculadora.

O processo para monitorizar uma página HTML é igual ao do Internet Explorer, no entanto não existe informação sobre as instâncias ativas de momento, por isso todo o trabalho efetuado pela biblioteca Microsoft Internet Controls necessita ser realizado na sua totalidade pela aplicação.

O procedimento consiste em primeiro lugar por procurar todas as instâncias deste *browser*; Em seguinte, para cada uma analisar todos os seus separadores; Por fim verificar o atributo "id"presente na *tag* "body"e em caso de correspondência iniciar a monitorização do processo.

Este procedimento é explicado em detalhe de seguida:

1. Obtém o elemento raiz que representa a janela principal do Windows ou por outras palavras, o ambiente do utilizador.

2. Os filhos representam todas as aplicações que estão abertas de momento, não incluindo aplicações em *background*. Portanto, de seguida é procurado em todos os filhos do elemento raiz por um elemento que tenha a propriedade "automationid"igual a "TitleBar". Poderia também ser encontrado pela classe "ApplicationFrameWindow", mas esta é partilhada por outras aplicações, como é o exemplo da calculadora. De momento não existe nenhuma outra aplicação a não ser o Microsoft Edge, filha do elemento raiz que partilhe este id, garantindo assim a inexistência de falhas neste ponto.

```
1  uiAutomation = AutomationElement.RootElement;
2
3  AutomationElementCollection edgebrowsers = uiAutomation.FindAll(TreeScope.Children,
4      new PropertyCondition(AutomationElement.AutomationIdProperty, "TitleBar"));
5
6  foreach (AutomationElement edge in edgebrowsers)
7  {
8      Program.Win32Callback childProc = new Program.Win32Callback((handle, pointer)
           =>
9      {
10         return EnumNewWindow(handle, pointer, session_id);
11     });
12     Program.EnumChildWindows((IntPtr)edge.Current.NativeWindowHandle, childProc, (
           IntPtr)0);
13 }
```

Listing 4.8: Pedaço de código executado após comunicação serviço a avisar que uma nova aplicação foi iniciada.

1. Agora existe um problema, porque o Microsoft Edge esconde desta *framework* os separadores que não estão ativos e a única solução passa por utilizar bibliotecas do Windows[5]. Para isso é importado a biblioteca "user32.dll"e as suas funções: EnumChildWindows, Win32Callback e GetWindowThreadProcessId.

2. Assim, com estas funcionalidades em conjunto, todos as janelas descendentes são expostas, sendo que todas aquelas que não sejam separadores são ignoradas. Isto é feito pela verificação do atributo "frameworkid"do elemento de automação, que no caso de um separador é identificado pelo valor "InternetExplorer".

3. Em seguida, após termos acesso ao objeto de automação do separador, apenas resta conseguir alcançar o seu documento (DOM) para verificar o atributo "id"da *tag* "body", que corresponde ao ID de sessão disponibilizado pelo serviço do Windows. Após analise da árvore, verificou-se que este elemento é o primeiro filho do separador.

[5]http://www.pinvoke.net/

4. Após correspondência do ID de sessão, é obtido o id do processo responsável por esse separador e a sua monitorização é desencadeada. É de salientar o retorno do booleano "false", na medida em que é obrigatório para parar a enumeração de janelas, não provocando tempo de processamento desnecessário após encontrar o separador correto.

```
1  [DllImport("user32.dll", SetLastError = true)]
2  public static extern uint GetWindowThreadProcessId(IntPtr hWnd, out uint
       lpdwProcessId);
3
4  [ComImport]
5  [Guid("00000114-0000-0000-C000-000000000046")]
6  [InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
7  public interface IOleWindow
8  {
9      void GetWindow(out IntPtr phwnd);
10     void ContextSensitiveHelp([In, MarshalAs(UnmanagedType.Bool)] bool fEnterMode);
11 }
12
13 public delegate bool Win32Callback(IntPtr hwnd, IntPtr lParam);
14 [DllImport("user32.Dll")]
15 [return: MarshalAs(UnmanagedType.Bool)]
16 public static extern bool EnumChildWindows(IntPtr parentHandle, Win32Callback
       callback, IntPtr lParam);
17
18 private bool EnumNewWindow(IntPtr handle, IntPtr pointer, string session_id)
19 {
20     try
21     {
22         AutomationElement elem = AutomationElement.FromHandle(handle);
23         if (elem.Current.FrameworkId == "InternetExplorer")
24         {
25             System.Windows.Automation.TreeWalker walker = System.Windows.Automation
                   .TreeWalker.RawViewWalker;
26             var doc = walker.GetFirstChild(elem); // Document Pane
27             string id = null;
28             if (doc != null)
29             {
30                 id = doc.Current.AutomationId;
31                 if (id == session_id)
32                 {
33                     uint pid;
34                     Program.GetWindowThreadProcessId(handle, out pid);
35                     websiteInstance website = new websiteInstance(session_id, pid);
36                     active_websites.Add(website);
37                     // Informs the server that this client is monitoring the
                           application with id and process with pid
38                     if (website.MonitorProcess() != -1)
```

```
39                      {
40                          Logger.Info("send message with proc id: " + pid);
41                          SendMessage("process: " + session_id + "|" + pid);
42                      }
43                      return false;
44                  }
45              }
46          }
47      }
48      catch (Exception e)
49      {
50          Logger.Error("PipeClient EnumNewWindow: " + e.Message);
51      }
52
53      return true;
54  }
```

Listing 4.9: Método responsável por enumerar todos os separadores do Microsoft Edge para encontrar e monitorizar o desejado.

Tal como acontece com o Internet Explorer, também neste sistema é garantida a consistência e minimização de erros por serem utilizadas bibliotecas do sistema. Além disso, obteve-se semelhante taxa de sucesso nos testes realizados.

4.1.4.5 Google Chrome

Atualmente o Google Chrome não expõe todos os seus elementos para a *framework* de automação, não permitindo aceder ao HTML ou DOM. No entanto existe um esforço por parte da comunidade para eliminar esta limitação, não só por conveniência dos programadores, mas também pelo facto de pessoas com limitações como é o caso dos invisuais, usarem aplicações que dependem desta *framework*. Uma outra abordagem, passa por associar um *listener* ao "Win32_ProcessStartTrace"para ter a informação de quando é criado ou removido um processo, neste caso "chrome.exe". Desde a versão 45 do Google Chrome, a sua gestão é feita da seguinte forma:

- É criado um processo por cada separador ou *plugin* carregado.
- Um processo extra é criado no momento que se escreve o URL da página, utilizado para executar o pré carregamento desta pelo *browser*
- No refrescamento de uma pagina, o processo do seu separador é eliminado e criado novamente.
- O processo é eliminado quando o utilizador fecha um separador ou quando um *plugin* já não é mais necessário.

Como grande parte dos elementos de automação não são expostos, apenas se pode saber o URL do separador ativo, para isso é utilizada uma função que está explicada de seguida.

```
1  private string GetChromeUrl(Process process)
2  {
3      try
4      {
5          if (process == null)
6              throw new ArgumentNullException("process");
7          if (process.MainWindowHandle == IntPtr.Zero)
8              return null;
9          AutomationElement element = AutomationElement.FromHandle(process.
               MainWindowHandle);
10         if (element == null)
11             return null;
12         AutomationElement edit = element.FindFirst(TreeScope.Descendants, new
               PropertyCondition(AutomationElement.ControlTypeProperty, ControlType.
               Edit));
13         string vp = ((ValuePattern)edit.GetCurrentPattern(ValuePattern.Pattern)).
               Current.Value as string;
14         return vp;
15     }
16     catch (Exception e)
17     {
18         Logger.Info("PipeClient GetChromeUrl method" + e.Message);
19     }
20     return "";
21 }
```

Listing 4.10: Função para obter o URL de um separador ativo do Google Chrome.

A funcao da Listing 4.10 procura pelo processo do Google Chrome que tem o "MainWindowHandle"diferente de 0, que corresponde ao processo do separador que está selecionado. Resta obter o elemento de automação e procurar nos seus descendentes pelo elemento do tipo "edit"que representa caixa de texto com o URL.

Como apenas existe a possibilidade de aceder ao separador ativo, esta abordagem não é uma opção, porque não se pode garantir que ao abrir vários separadores com a mesma aplicação, estas sejam monitorizadas corretamente. No entanto é um pequeno avanço, porque a partir do momento em que este *browser* forneça a informação sobre os seus elementos como o Internet Explorer, Microsoft Edge ou mesmo Mozilla Firefox, esta técnica será facilmente adaptada para obter os resultados desejados.

4.1.5 Serviço Web - JSPCREST

Para implementar um serviço web, podem ser utilizadas duas arquiteturas: REST e SOAP. Optou-se para esta situação, pela abordagem REST por ser mais escalável e por suportar comunicações com dados em JSON, texto e XML, ao contrário da arquitetura SOAP que restringe à comunicação apenas de dados em XML.

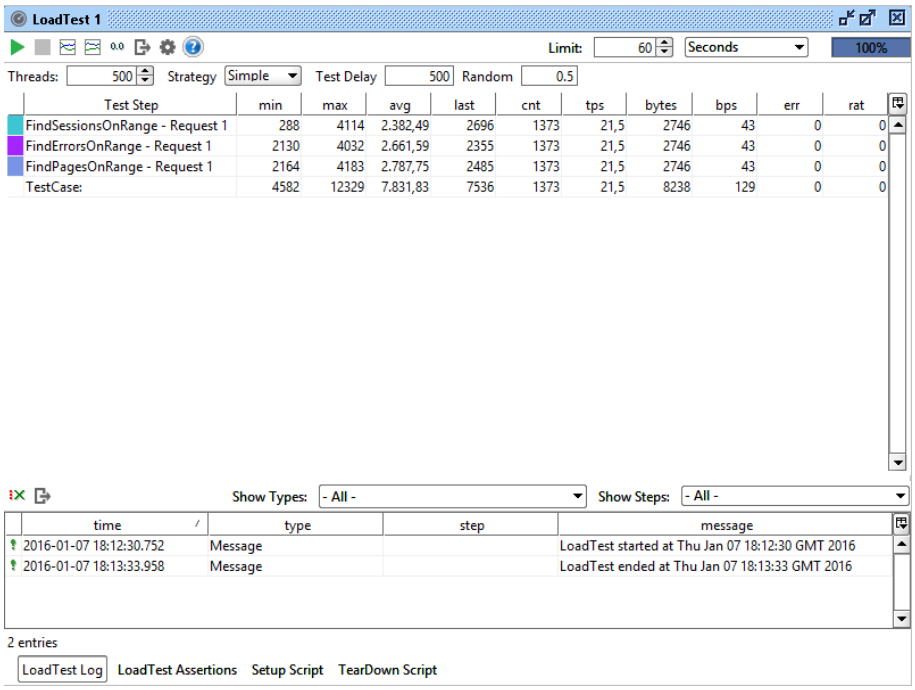

Figura 4.6: Árvore de Automação do Windows.

Também para este serviço foram executados testes de stress, onde são realizados 1500 pedidos ao servidor a cada 500 milissegundos, não existindo qualquer falha de comunicação. Relativamente ao tempo de resposta, tendo em atenção que também está a ser transferida informação neste caso, os resultados obtidos são bastante satisfatórios.

4.1.6 Aplicação Web - JSPCDashboard

O último componente é onde todo o esforço resultante desta dissertação é representado. A nível empresarial, representa um único local onde se pode visualizar as milhares ou até milhões de interações de todos os seus clientes.

Com a quantidade de informação que é armazenada, as possibilidades de representação e organização dos dados é imensa. Tendo em conta a extensão desta dissertação e o tempo cedido, sem contar com a sua escrita e a documentação necessária, foram desenvolvidos apenas alguns ecrãs que serão explicados nas próximas secções.

Figura 4.7: Ecrã principal com o menu de navegação aberto.

4.1.6.1 Visualização de Sessões

A página inicial da Dashboard, apresenta imediatamente ao utilizador todas as sessões do dia, em que cada uma tem um pequeno sumário do número e tipo de interações que ocorreram nesta. Aqui podemos alterar a data de visualização através do calendário, selecionando uma opção já predefinida ou então, selecionando uma data exata ao nosso gosto. Neste último caso, também se pode selecionar o tempo até uma resolução de 30 minutos.

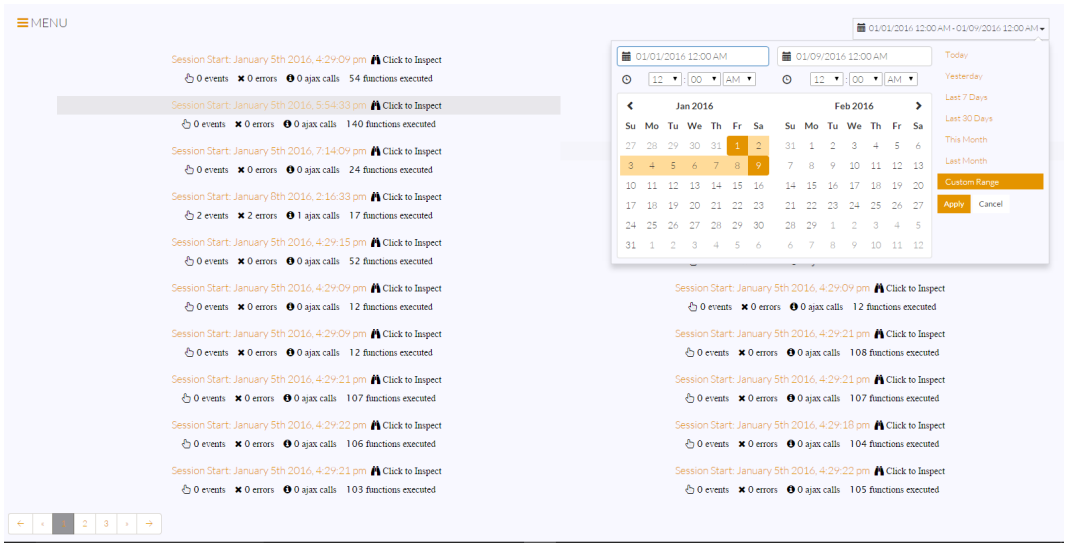

Figura 4.8: Ecrã de visualização de sessões.

São apresentadas no máximo 20 sessões por página, mas existe a opção de navegação no canto inferior esquerdo, no caso do número de sessões ser superior a este valor.

Ao clicar num item de sessão, o utilizador é direcionado para o ecrã da sua analise individual.

4.1.6.2 Analise de uma Sessão

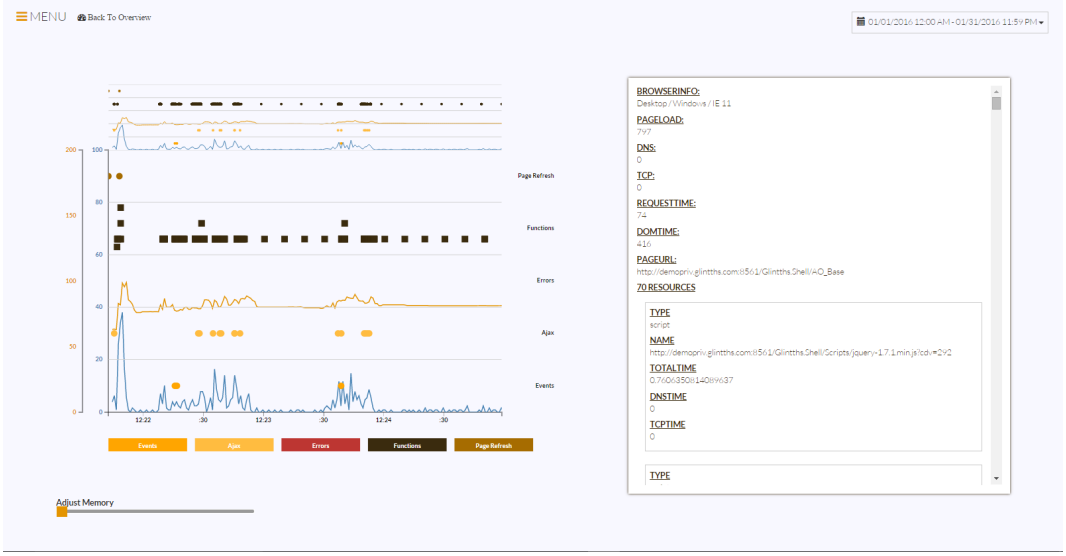

Figura 4.9: Ecrã de analise de uma sessão - Teste da aplicação Gestão de Encaminhamentos da Glintt HS.

Na Figura 4.9 está visível o gráfico de uma sessão. Neste gráfico é representada uma *timeline* desde o início da sessão até ao seu fim, com a utilização de memória e processador e também 5 níveis em que cada um representa um tipo de interação que pode ser: refrescamento ou carregamento de uma página, execução de uma função, erro, pedido Ajax ou evento. Por baixo do gráfico estão presentes na cor respetiva retângulos com o tipo de interação. Clicando nestes, o seu nível desaparece, aparecendo ao clicar novamente, o que permite analisar interações em conjunto ou individualmente. Mais abaixo está também presente uma barra ajustável para alterar os limites de memória, estes limites variam entre 200MB até um total de 2048 MB.

Do lado direito é o local onde estão apresentados os detalhes de uma interação e para os aceder, basta clicar no elemento que se deseja analisar.

No caso de uma função, existe um botão na área de detalhe que apresenta a quem está analisar, um editor com a função que foi executada, apresentando automaticamente todos os erros e problemas existentes. O utilizador pode alterar em tempo real o código no editor para os corrigir. Em baixo está apresentada uma imagem com um exemplo prático.

Figura 4.10: Ecrã de exploração de uma função executada.

4.1.6.3 Páginas Visitadas

Aqui, é apresentado ao utilizador um gráfico de barras em que cada uma representa uma determinada página e o seu valor o número de visitas realizadas no espaço de tempo selecionado.

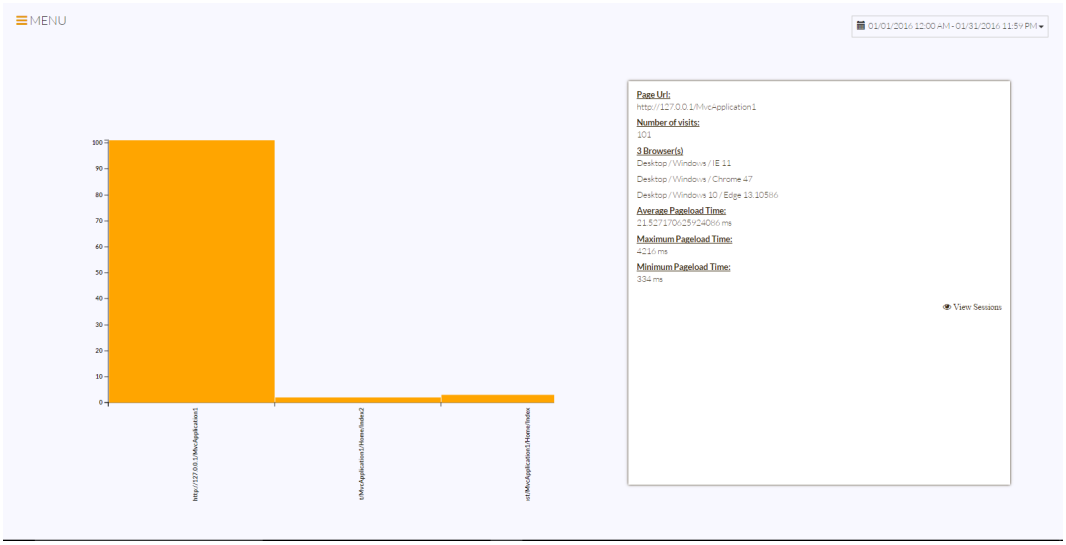

Figura 4.11: Ecrã de páginas visitadas.

Ao clicar em uma das barras do gráfico é descrita a seguinte informação:

Page URL Representa o link da página cuja barra do gráfico foi selecionada.

Number of Visits Número de visitas a esta aplicação por parte dos clientes.

Browsers Tipos de browsers que visitaram a aplicação.

Average Pageload Time Tempo médio do carregamento da página.

Maximum Pageload Time Tempo máximo de carregamento da página.

Minimum Pageload Time Tempo mínimo de carregamento da página.

Esta informação permite-nos identificar quais as aplicações ou páginas de uma aplicação são mais visitadas. Mesmo que uma página tenha problemas de performance, se o número de visitas é reduzido, o esforço necessário para os resolver pode não ser compensatório. Também é dado ao utilizador a possibilidade de ver todas as sessões onde esta página foi visitada, podendo analisar individualmente cada uma delas. Para isso basta clicar em "Inspect Sessions"e de seguida é apresentada toda a lista de sessões no mesmo formato do visível na Figura 4.8.

4.1.6.4 Sumário Ajax

Neste local o utilizador tem acesso a todos os pedidos AJAX em um determinado espaço de tempo. Estes pedidos estão agregados por URL da chamada e código de retorno. Como é visível na Figura 4.12, existem 2 tipos de pedidos com o mesmo URL: um com o código de retorno 200 e outro com o código 404. A sua visualização é feita através do "click"no elemento respetivo da lista.

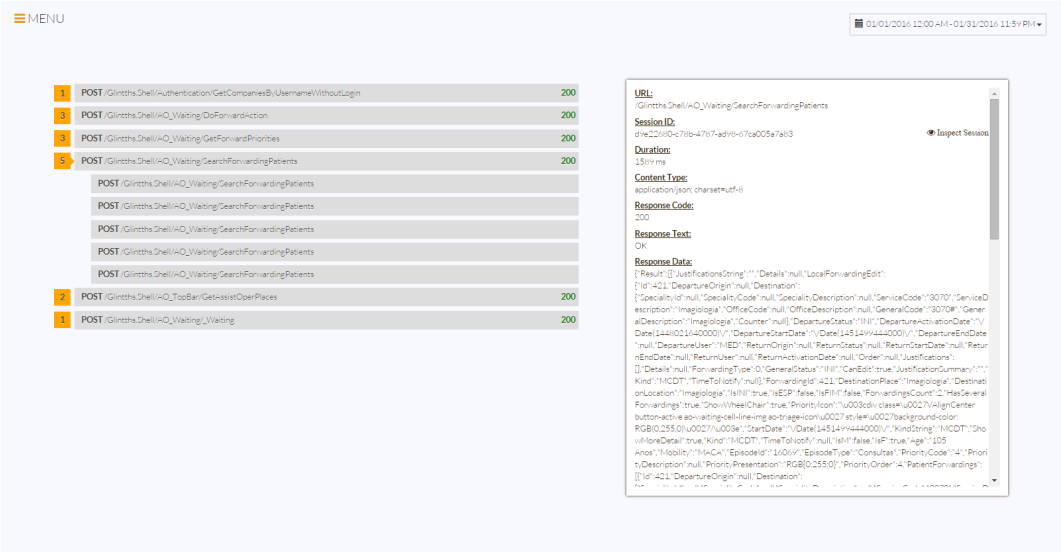

Figura 4.12: Ecrã dos pedidos AJAX - Teste da aplicação Gestão de Encaminhamentos da Glintt HS.

A informação do pedido AJAX é a seguinte:

URL Representa o link do servidor do pedido.

Session ID Aqui é apresentado o ID da sessão onde foi realizado o pedido AJAX. Ao selecionar a opção "Inspect Session", o utilizador é redirecionado para a analise individual dessa sessão.

Duration Tempo total de duração do pedido.

Content Type Formato do conteudo enviado para o servidor, como por exemplo, "applcation/json", "application/x-www-form-urlencoded", "plain/text", entre outros.

Response Code Código de resposta enviado pelo servidor. [6]

Response Text Texto de resposta do servidor.

Response Data Dados enviados pelo servidor para o *browser*.

4.1.6.5 Ocorrência de Erros

O propósito da visualização de erros é proporcionar à empresa um local para retirar imediatamente a informação, sobre todos os erros que estão a acontecer durante a utilização das suas aplicações e quais os mais frequentes.

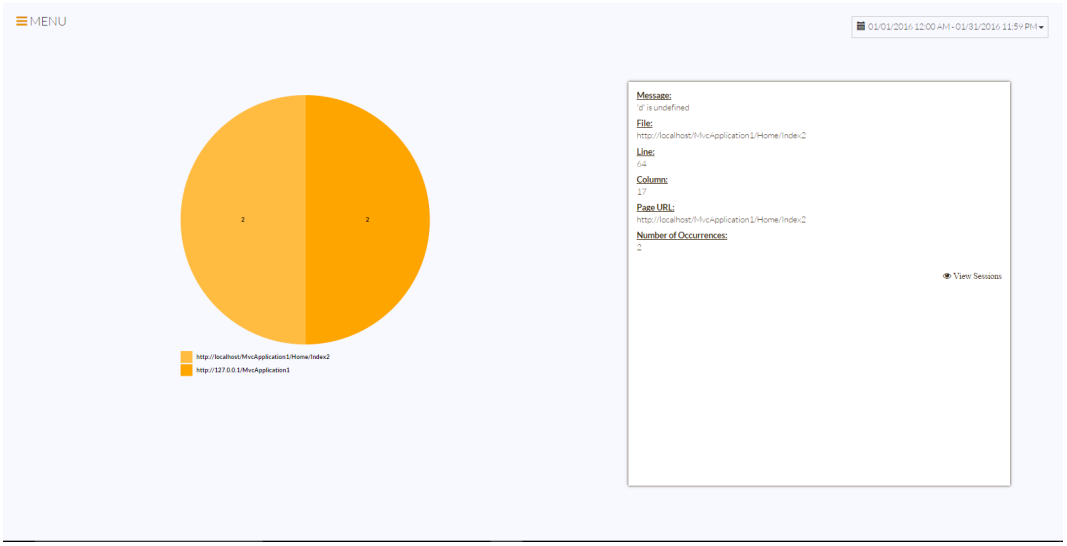

Figura 4.13: Ecrã de ocorrências de erros.

Para identificar e resolver um erro rapidamente é apresentada informação não só sobre a mensagem de erro, mas também, sobre o ficheiro nomeadamente o nome, a linha e a coluna. Caso o erro ocorra em código incluído diretamente na página e não em um recurso carregado o nome do ficheiro toma o nome do URL da página.

4.1.6.6 Analise de Código

A análise de código Javascript é um dos tópicos principais desta dissertação. O objetivo da empresa é avaliar o código das suas aplicações em utilização e também certificar as suas bibliotecas.

Apesar da análise de código dinâmica ser realizada na inspeção individual de uma sessão onde é possível verificar todas as funções executadas, existia ainda a necessidade de certificar o código

[6] https://www.w3.org/Protocols/HTTP/HTRESP.html

de ficheiros Javascript. Para isso foi adicionada uma página à Dashboard onde este processo é realizado. O utilizador tem a opção de fazer o upload de ficheiros ou pastas diretamente para os servidor, sendo apresentada de imediato a sua árvore de ficheiros.

A árvore de ficheiros é gerada por um *plugin* de JQuery designado de "JQuery File Tree". Este *plugin* utiliza um conector que faz a ligação entre o servidor e o cliente através de linguagem Visual Basic e foi necessário adaptar o mesmo para acionar a seleção de ficheiros.

Além disto, foi também embutido na página um editor de texto para que seja possível introduzir código. Estes dois componentes estão visíveis na Figura 4.16.

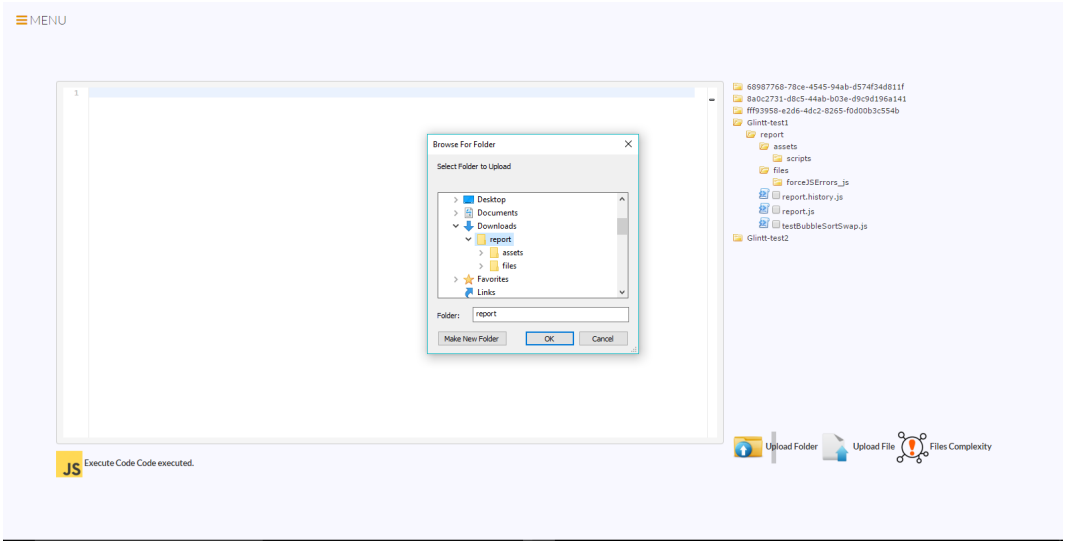

Figura 4.14: Ecrã de análise de código - Upload de uma pasta.

O código ao ser introduzido no editor, seja por escrita direta ou por cópia, é imediatamente avaliado através do *parsing* do mesmo pelo Esprima e respetiva análise da árvore sintática. Ao utilizador são apresentados todos os erros, possíveis *leaks* de variáveis globais, variáveis não definidas, entre outros. Para além de todas as inconsistências analisadas automaticamente, existe também a opção de executar o código ao selecionar a opção "Execute Code". Se não existirem erros, este é processado e é adicionado ao editor um símbolo na linha de cada função, com o número de vezes que esta é executada.

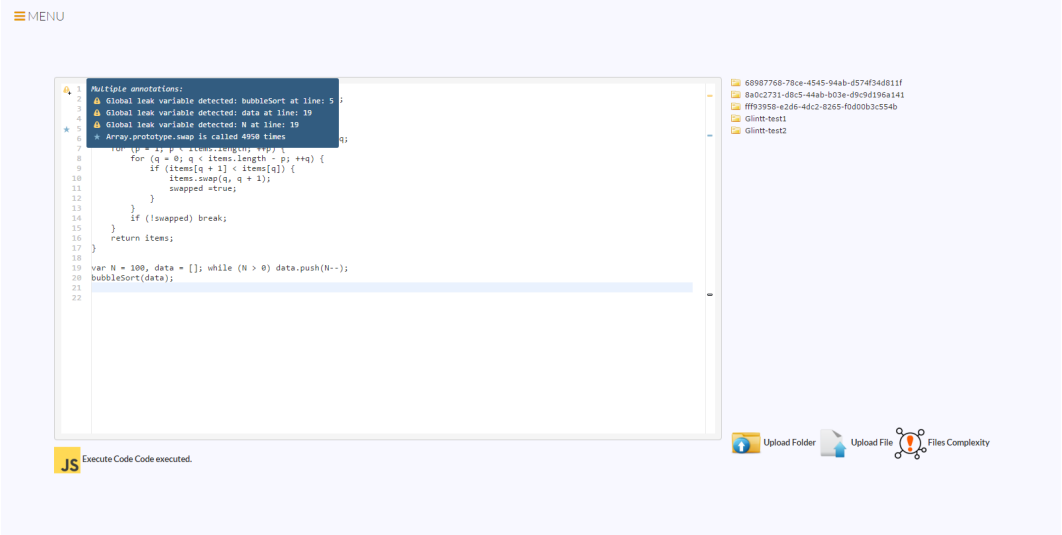

Figura 4.15: Ecrã de análise de código - Informação sobre a execução de código.

O próximo passo na certificação de código Javascript é a análise de complexidade de um ficheiro. Para atingir tais resultados é utilizado o módulo "Plato"de NodeJS que gera um relatório completo de todos os ficheiros selecionados, com a seguinte informação:

Summary Um sumário de todos os ficheiros que inclui a média de linhas de código e a média da facilidade de manutenção.

Maintainability Apresenta um gráfico de barras em que cada uma representa um ficheiro diferente. O seu valor varia entre 0 e 100, sendo que valores maiores representam uma maior facilidade de manutenção do código.

Lines of code Gráfico de barras com o total de linhas de código por ficheiro - SLOC (Source Lines of Code).

Estimated errors in implementation Número de erros possíveis por ficheiro.

Lint errors Número de erros por ficheiro.

Files Lista de ficheiros analisados.

Este relatório não pode ser gerado no lado do cliente, pois seria necessário instalar uma aplicação no seu dispositivo para executar o módulo através da linguagem *Server-Side* NodeJS.

Esta restrição foi ultrapassada, através da introdução do "IISNode"que permite hospedar aplicações NodeJS, no servidor IIS utilizado pela .Net Framework. A Dashboard, sendo uma aplicaçao Asp.Net Mvc, necessita de um pedaço de código para incluir este módulo. O código está explicito na Listing 4.11.

```
1  <location path="Node">
2    <system.webServer>
```

```
3       <handlers>
4         <add name="iisnode" path="index.js" verb="*" modules="iisnode" />
5       </handlers>
6     </system.webServer>
7   </location>
```

Listing 4.11: Código incluido no Web Config para indicar que o ficheiro "index.js"da pasta "Node"necessita de NodeJS para executar.

Para realizar a análise de complexidade de ficheiros estes têm obrigatoriamente de estar presentes no servidor ou seja, o utilizador deve fazer o seu upload individual ou do seu diretório, através das funcionalidades explicadas anteriormente. Após a sua amostragem na árvore de ficheiros, o próximo passo é selecionar o que se deseja inspecionar.

No caso de um diretório completo este processo poderia ser fastidioso, por isso adicionou-se a funcionalidade em que ao selecionar um diretório da árvore, todos os seus sub-diretórios e ficheiros são automaticamente selecionados. Convém também referir que ao clicar novamente em um diretório que está aberto, este é fechado e por consequência todos os seus descendentes ficam descartados.

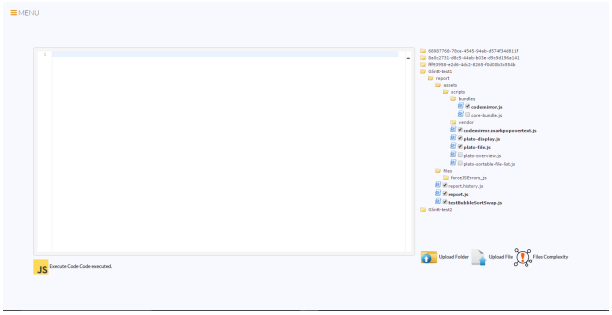

Figura 4.16: Seleção de ficheiros para realizar a sua análise de complexidade.

Após o utilizador executar a opção "Files Complexity"é enviado para o servidor a informação do caminho de todos os ficheiros que foram selecionados. De seguida, o caminho de todos estes é armazenado em um ficheiro de texto. Ao receber a resposta do servidor que a informação foi armazenada com sucesso, é iniciado um novo separador do *browser* com o URL da aplicação de NodeJS. Esta aplicação gera uma pasta com o relatório de todos os ficheiros, cujo caminho está presente no ficheiro previamente gerado. Por fim, esta mesma aplicação cria um servidor HTTP em que o seu conteúdo é a página HTML do relatório, gerada pelo módulo Plato.

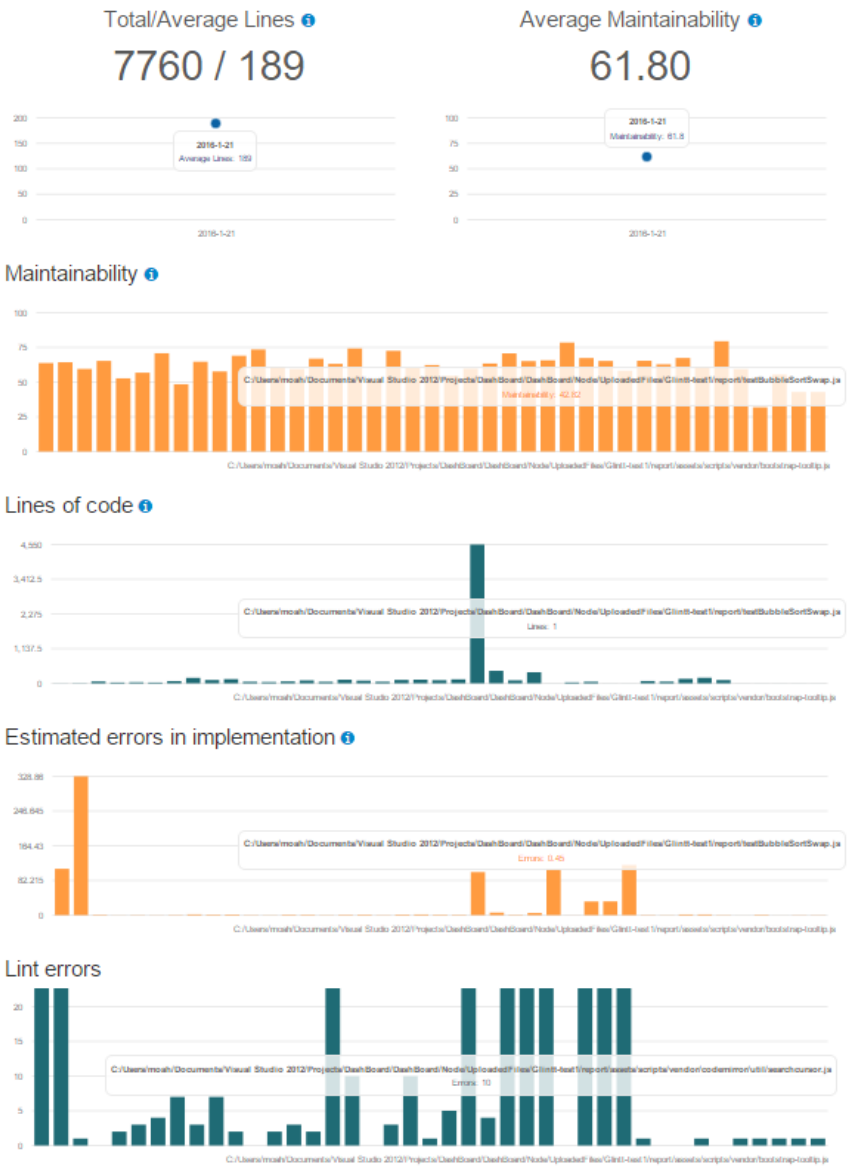

Figura 4.17: Relatório de complexidade de uma pasta com ficheiros Javascript.

Nas Figuras 4.17 e 4.18, está visível a página HTML de um relatório gerado pelo Plato de uma pasta com 41 ficheiros Javascript.

Esta informação é armazenada no formato JSON e de futuro estes dados podem ser organizados e apresentados de uma forma diferente, caso a empresa o deseje.

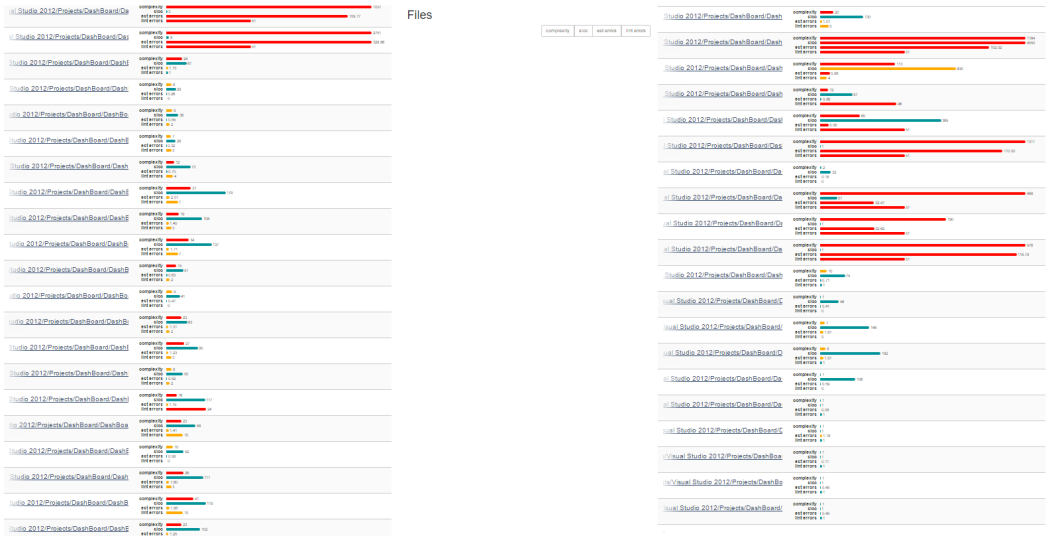

Figura 4.18: Lista dos ficheiros da pasta inspecionada.

A partir daqui fica fácil verificar onde poderá existir o maior número de inconsistências.

Capítulo 5

Conclusões e Trabalho Futuro

Neste capitulo são exibidas as conclusões retiradas desta Dissertação.

5.1 Satisfação dos Objetivos

No início do desenvolvimento não existia conhecimento se era possível implementar a ideia proposta na sua totalidade, não só pela sua amplitude, mas também por não existir algo na área sobre alguns dos tópicos.

Todo o trabalho de investigação e desenvolvimento que foi realizado protagonizou um enriquecimento benéfico tanto a nível do Sistema Operativo Windows, como na gestão e estrutura dos principais *browsers* do mercado.

Todos os requisitos foram cumpridos e os resultados obtidos são extremamente satisfatórios, na medida em que se conseguiu comprovar o conceito, com um modelo prático muito similar ao proposto pela empresa. Com estes resultados e todos os testes realizados, pode-se afirmar que este produto está preparado para ser testado com os clientes e em larga escala.

5.2 Futuro do JSPCertifier

A área de *web profiling* está em crescimento, o que faz com que esta solução possa ter um futuro promissor. Como existe uma constante alteração e introdução de novas funcionalidades, a arquitetura da solução foi pensada de modo a que exista facilidade em adicionar, remover ou alterar os seus módulos.

Futuramente o objetivo passará por fornecer suporte para os restantes *browsers*, dispositivos móveis e também por aproveitar toda a informação recolhida. Será necessário criar novas funcionalidades tanto na representação de resultados, como na análise de código para que com mais facilidade se possam identificar os problemas que surgem ao longo do tempo, porque nunca nenhuma aplicação está perfeita, há sempre margem para melhorar.

Esta solução proporciona uma vantagem competitiva a longo prazo, visto que a empresa poderá melhorar e certificar constantemente as suas aplicações, conseguindo satisfazer cada vez mais os seus clientes.

Referências

[AZGvG09] Rui Abreu, Peter Zoeteweij, Rob Golsteijn e Arjan J C van Gemund. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software*, 82(11):1780–1792, 2009. doi:10.1016/j.jss.2009.06.035.

[AZV07] Rui Abreu, Peter Zoeteweij e Arjan J C Van Gemund. On the accuracy of spectrum-based fault localization. *Proceedings - Testing: Academic and Industrial Conference Practice and Research Techniques, TAIC PART-Mutation 2007*, pages 89–98, 2007. doi:10.1109/TAICPART.2007.4344104.

[Boe12] J Boekesteijn. *JavaScript Code Quality Analysis*. PhD thesis, TU Delft, Delft University of Technology, 2012.

[Bru09] Jake Brutlag. Speed matters for google web search. *Google. June*, page 2009, 2009.

[CAFD13] Jose Campos, Rui Abreu, Gordon Fraser e Marcelo D'Amorim. Entropy-based test generation for improved fault localization. *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013 - Proceedings*, pages 257–267, 2013. doi:10.1109/ASE.2013.6693085.

[Deva] Google Developers. Evaluating network performance.

[Devb] Google Developers. Javascript memory profiling.

[dSN14] André da Silva Nogueira. André da Silva Nogueira Profiling de aplicações Web : Estudo comparativo entre aplicações Java Web e aplicações RoR. 2014.

[GCA13] Carlos Gouveia, José Campos e Rui Abreu. Using HTML5 visualizations in software fault localization. *2013 1st IEEE Working Conference on Software Visualization - Proceedings of VISSOFT 2013*, 2013. doi:10.1109/VISSOFT.2013.6650539.

[Gem] Arjan J C Van Gemund. An Evaluation of Similarity Coefficients for Software Fault Localization.pdf.

[GG14a] Pablo Garaizar e Mariluz Guenaga. A multimodal learning analytics view of HTML5 APIs. *Proceedings of the Second International Conference on Technological Ecosystems for Enhancing Multiculturality - TEEM '14*, (OCTOBER 2014):275–281, 2014. doi:10.1145/2669711.2669911.

[GG14b] Bhanwar Gupta e Puneet Goswami. Memory leaks in Web 2 . 0 Applications. 106(May):101–106, 2014.

[GHMP02] Dennis F Galletta, Raymond M Henry, Scott Mccoy e Peter Polak. Web site delays: How tolerant are users? *Information Systems Research*, 17(December 2002):20–37, 2002. doi:http://melody.syr.edu/hci/jais04/JAIS_Galletta.pdf.

[Gri13] Ilya Grigorik. Measuring network performance with resource timing api. December 11 2013.

[Gri14] Ilya Grigorik. Measuring the critical rendering path with navigation timing. April 1 2014.

[Hid] Ariya Hidayat. Esprima parser.

[(In10] Andy Idsinga (Intel). Intel cpu web api documentation and examples. April 8 2010.

[int11] ECMA international. Ecmascript® language specification. June 2011.

[Iri12] Paul Irish. When milliseconds are not enough: performance.now. August 16 2012.

[Kie10] Holger M. Kienle. It's about Time to Take JavaScript (More) Seriously. *IEEE Software*, 27(3):60–62, 2010. doi:10.1109/MS.2010.76.

[KNB+05] Markus Kobel, Oscar Nierstrasz, Horst Bunke, Tudor Gırba e Michele Lanza. Parsing by example. *Institut fur Informatik und angewandte Mathematik*, 2005.

[Lee11] Wim Leers. Web performance optimization: Analytics. 2011.

[Mee13] Patrick Meenan. How Fast is Your Web Site? *Queue - Mobile Web Development*, 11(2):1–11, 2013. doi:10.1145/2436696.2446236.

[Nag13] ZSOLT Nagy. Improved speed on intelligent web sites. *Recent Advances in Computer Science, Rhodes Island, Greece*, pages 215–220, 2013.

[Net] Mozilla Developer Network. Parser api.

[Net15a] Mozilla Developer Network. Memory management. April 11 2015.

[Net15b] Mozilla Developer Network. Navigation timing. May 8 2015.

[Nic13] Alex Nicolaou. Best practices on the move: building web apps for mobile devices. *Queue*, 11(6):30, 2013.

[O'r07] Tim O'reilly. What is web 2.0: Design patterns and business models for the next generation of software. *Communications & strategies*, (1):17, 2007.

[Rem15] Garret Rempel. Defining standards for web page performance in business applications. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, pages 245–252. ACM, 2015.

[RGEV11] Gregor Richards, Andreas Gal, Brendan Eich e Jan Vitek. Automated construction of JavaScript benchmarks. *ACM SIGPLAN Notices*, 46(10):677, 2011. doi:10.1145/2076021.2048119.

[RLBV10] Gregor Richards, Sylvain Lebresne, Brian Burg e Jan Vitek. An analysis of the dynamic behavior of JavaScript programs. *ACM SIGPLAN Notices*, 45(6):1, 2010. doi:10.1145/1809028.1806598.

[SB01] Peter Sevcik e John Bartlett. Understanding web performance. *Business Communications Review*, 31(10):28–36, 2001.

[Sou08] Steve Souders. High-performance web sites. *Communications of the ACM*, 51(12):36, 2008. doi:10.1145/1409360.1409374.

[SYA+14] a D D Split, T O Your, W E B App, Intro To e F O R Web. Web development. 2014.

[W3C13] W3C. Navigation timing. January 25th 2013.

[W3C15] W3C. Resource timing. June 4 2015.

[Wil15a] Alan Cameron Wills. Api do application insights para métricas e eventos personalizados. Oct 23 2015.

[Wil15b] Alan Cameron Wills. Application insights for javascript web apps. Nov 17 2015.

[Wil15c] Alan Cameron Wills. Export telemetry from application insights. Nov 15 2015.

[Wil15d] Alan Cameron Wills. What is application insights? Nov 23 2015.

[XZZ12] Wei Xu, Fangfang Zhang e Sencun Zhu. The power of obfuscation techniques in malicious JavaScript code: A measurement study. *Proceedings of the 2012 7th International Conference on Malicious and Unwanted Software, Malware 2012*, pages 9–16, 2012. doi:10.1109/MALWARE.2012.6461002.

[YIHU13] Waheed Yasin, Hamidah Ibrahim, Nor Asilah Wati Abdul Hamid e Nur Izura Udzir. The affects of caching in browser stage on the performance of web items delivery. In *The Second International Conference on Digital Enterprise and Information Systems (DEIS2013)*, pages 212–219. The Society of Digital Information and Wireless Communication, 2013.

[YL12] Mj Yuan e J Long. CloudBees: A Resource Guide for Teaching Clouding Computing on a Java Platform. *... of the Information Systems Educators Conference ...*, pages 1–7, 2012.